Integracija generativnog programiranja i skriptnih jezika

Radošević, Danijel

Uvod

1 UVOD 1

1.1 CILJEVI RADA 1
1.2 HIPOTEZE 2
1.3 POSTIGNUTI REZULTATI 2
1.4 STRUKTURA RADA 4

2 AKTUALNE TEHNOLOGIJE I TRENDOVI NA PODRUČJU PROGRAMIRANJA 5

2.1 RAZVOJ PROGRAMSKIH JEZIKA 5
2.1.1 Početni programski jezici 8
2.1.2 Strukturni pristup programiranju 9
2.2 OBJEKTNO ORIJENTIRANI PRISTUP PROGRAMIRANJU 11
2.2.1 Trendovi razvoja objektno orijentiranog programiranja u proteklom desetljeću 12
2.2.2 Objektno modeliranje 14
2.2.2.1 UML 14
2.2.2.2 Generativni domenski model 16
2.2.2.3 Modeliranje u okviru aspektno-orijentiranog programiranja 16
2.3 JEZICI SKRIPATA 18
2.3.1 Slabosti objektnog pristupa 18
2.3.2 Primjena jezika skripata 20
2.3.2.1 Područja na kojima jezici skripata imaju široku primjenu 20
2.3.3 Jezici bliski jezicima skripata 21
2.3.3.1 C# 21
2.3.4 Slabosti jezika skripata 23
2.3.5 Očekivanja od primjene jezika skripata u okviru generativnog programiranja 24
2.4 GENERATIVNO PROGRAMIRANJE 24
2.4.1 Temeljne discipline Generativnog programiranja 27
2.4.1.1 Metaprogramiranje 27
2.4.1.1.1 Ilustrativni primjer 28
2.4.1.2 Generičko programiranje 31
2.4.1.3 Objektno orijentirano programiranje 31
2.4.1.4 Aspektno orijentirano programiranje 32
2.4.1.5 Domenski inžinjering 33
2.4.2 Aktivne biblioteke 34

3 DOSADAŠNJI PRIMJERI UPOTREBE JEZIKA SKRIPATA U GENERATIVNOM PROGRAMIRANJU 35

3.1 OPEN PROMOL 35
3.1.1 Glavni koncepti Open Promola 36
3.1.2 Osnovna svojstva sintakse i semantike Open Promola 36
3.1.3 Primjeri 37
3.2 CODEWORKER 38
3.2.1 Svojstva jezika 39
3.2.2 Primjer 39
3.3 SVOJSTVA DOSADAŠNJIH PRIMJERA UPOTREBE JEZIKA SKRIPATA U OKVIRU GENERATIVNOG PROGRAMIRANJA 40

4 SKRIPTNI MODEL GENERATORA APLIKACIJA 41

4.1 RAZINE GENERIRANJA 42
4.2 DIJAGRAMI SKRIPTNOG MODELA 44
4.2.1 Dijagram parametara opisa aplikacije 44
4.2.2 Dijagram metaskripata 45
4.2.2.1 Raspored elemenata dijagrama metaskripata 47
4.3 GENERATORI PREMA SKRIPTNOM MODELU 49
4.3.1 Jednostavan generator 49
4.3.1.1 Skriptni model jednostavnog generatora 52

4.3.2 Višerazinski generator *54*
4.3.2.1 Skriptni model višerazinskog generatora 55
4.3.2.2 Primjer: razvoj jednostavnog generatora aplikacija u C++ 57
4.3.2.2.1 Izrada višerazinskog generatora prema skriptnom modelu 57
4.3.3 Parametrizirani višerazinski generator *61*
4.3.3.1 Funkcije generiranja prema skriptnom modelu 61
4.3.3.2 Gradivni elementi višerazinskog parametriziranog generatora 63
4.3.3.3 Svojstva višerazinskog parametriziranog generatora 66
4.3.4 Automatizacija izrade generatora *67*
4.3.5 Metagenerator *69*
4.3.5.1 Odjeljak za učitavanje predložaka (metaskripata) koje će koristiti generator: 69
4.3.5.2 Odjeljak za učitavanje opisa aplikacije koji će koristiti generator: 70
4.3.5.3 Odjeljak za učitavanje opisa aplikacije 70
4.3.5.4 Odjeljak za generiranje aplikacije 70
4.3.5.5 Odjeljak za spremanje aplikacije 72
4.3.5.6 Korištenje komentara 72
4.3.6 Primjer: razvoj jednostavnog generatora aplikacija uz pomoć metageneratora *73*
4.3.6.1 Tekstualni opis generatora 73
4.3.6.2 Izrađeni generator 74
4.3.6.3 Primjer generiranja aplikacije u C++ 75
4.3.6.3.1 Opis aplikacije 75
4.3.6.3.2 Generirani kod aplikacije 75
4.4 GRAMATIKA SKRIPTNOG MODELA 76
4.4.1 Gramatika tekstualnog opisa aplikacije *77*
4.4.2 Gramatika dijagrama parametara opisa aplikacije *77*
4.4.3 Gramatika dijagrama metaskripata *77*
4.4.4 Gramatika tekstualnog opisa generatora *77*
4.5 ODNOS OBJEKTNOG I SKRIPTNOG MODELA 78
4.5.1 Klasa - metaskripta *78*
4.5.2 Objekt - skripta *79*
4.5.3 Enkapsulacija *79*
4.5.4 Nasljeđivanje *80*
4.5.5 Dijagram slučajeva upotrebe - dijagram parametara opisa aplikacije *81*
4.5.6 Dijagram suradnje - dijagram metaskripata *81*
4.5.7 Dijagram klasa - dijagram metaskripata *82*

5 GENERATIVNI RAZVOJ APLIKACIJA 86

5.1 PREGLED DO SADA RAZVIJENIH GENERATORA 87
5.1.1 Generator web aplikacija za daljinsko održavanje baza podataka *87*
5.1.1.1 Skriptni model generatora aplikacija za daljinsko održavanje baza podataka 88
5.1.1.2 Primjer generirane aplikacije za daljinsko održavanje baza podataka 92
5.1.2 Generator korisničkih web aplikacija *94*
5.1.2.1 Skriptni model generatora korisničkih web aplikacija 95
5.1.2.2 Primjer generirane korisničke web aplikacije 97
5.1.3 Generator anketnih upitnika *99*
5.1.3.1 Skriptni model generatora anketnih upitnika 99
5.1.3.2 Primjer generiranog anketnog upitnika 102
5.2 MODIFIKACIJE APLIKACIJA I GENERATORA 104
5.2.1 Modifikacije opisa aplikacije *105*
5.2.2 Modifikacije metaskripata *106*
5.2.3 Modifikacije generatora *106*
5.2.3.1 Generator web upitnika za testiranje 107

6 PROJEKT PRIMJENE GENERATIVNOG RAZVOJA APLIKACIJA 110

6.1 PROJEKTNI ZADATAK: SUSTAV ZA ANKETIRANJE GOSTIJU HOTELA 110
6.1.1 Zahtijevane funkcionalnosti sustava *110*
6.1.1.1 Održavanje baze podataka o gostima 111
6.1.1.2 Slanje e-mail poruka gostima 112
6.1.1.3 Anketiranje gostiju 113
6.1.1.4 Prikaz i obrada rezultata ankete 115
6.1.2 Generativni razvoj sustava *115*
6.1.2.1 Strojni i programski elementi sustava za anketiranje gostiju 115
6.1.2.2 Primijenjeni generatori 116

6.1.2.3 Održavanje baze podataka o gostima i baze predložaka ... 116
6.1.2.4 Slanje e-mail poruka gostima i učitavanje podataka o gostima ... 118
6.1.2.5 Anketiranje gostiju ... 120
6.1.2.6 Prikaz i obrada rezultata ankete ... 126
6.1.2.6.1 Modifikacije generatora anketnih upitnika ... 126
6.2 IZRAĐENI SUSTAV ZA ANKETIRANJE GOSTIJU HOTELA ... 127
6.2.1 Održavanje baze podataka o gostima i baze predložaka ... 128
6.2.2 Učitavanje podataka o gostima i slanje e-mail poruka ... 129
6.2.3 Anketiranje gostiju ... 131
6.2.4 Prikaz i obrada rezultata ankete ... 132
6.3 PREDNOSTI GENERATIVNOG RAZVOJA APLIKACIJA ... 134

7 TESTIRANJE PRIMJENE KONCEPTA GENERATIVNOG PROGRAMIRANJA TEMELJENOG NA JEZICIMA SKRIPATA ... 135

7.1 SKRAĆENJE RAZVOJNOG CIKLUSA APLIKACIJA ... 135
7.1.1 Testovi razine jezika ... 135
7.1.1.1 Omjer veličine programskog koda opisa aplikacija i odgovarajućih generiranih aplikacija. 136
7.1.1.1.1 Aplikacija za anketiranje gostiju hotela ... 136
7.1.1.1.2 Aplikacija za održavanje baze ispitnih rokova i prezentaciju ispitnih rokova ... 137
7.1.1.1.3 Anketni upitnik ... 137
7.1.1.1.4 Jednostavan web portal ... 137
7.1.1.1.5 Aplikacija za online testiranje korištenjem Spitzbergovih upitnika ... 138
7.1.1.1.6 Web testovi za C++ ... 138
7.1.1.1.7 Očekivani omjeri veličine opisa aplikacija i generiranog programskog koda ... 139
7.2 OPTIMIZACIJA PERFORMANSI IZRAĐENIH APLIKACIJA ... 140
7.2.1 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora aplikacija za daljinsko održavanje baza podataka ... 141
7.2.2 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora korisničkih web aplikacija ... 145
7.2.3 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora web upitnika 149
7.3 POJEDNOSTAVLJENJE ODRŽAVANJA APLIKACIJA ... 152
7.3.1 Pojednostavljenje održavanja aplikacija na razini modifikacija opisa aplikacije ... 152
7.3.2 Pojednostavljenje održavanja aplikacija na razini modifikacija predložaka programskog koda 153
7.3.3 Pojednostavljenje održavanja aplikacija na razini modifikacija generatora ... 154

8 ZAKLJUČAK ... 155

LITERATURA ... 159

PRILOG A: POJMOVNIK GENERATIVNOG PROGRAMIRANJA PREMA KONCEPTU TEMELJENOM NA JEZICIMA SKRIPATA ... 164

1 UVOD

Generativno programiranje je nova znanstvena disciplina, budući da su se prvi radovi pojavili tek potkraj 1990.-tih godina. Prema definiciji, generativno programiranje predstavlja "...dizajniranje i implementaciju programskih modula koji se mogu kombinirati radi generiranja visoko specijaliziranih i optimiziranih sustava koji omogućuju rješavanje specifičnih zadataka." (Eisenecker, 1997. str 2). Osnovni cilj je povećanje produktivnosti programera i brži razvoj aplikacija korištenjem odgovarajućih generatora. To područje postaje sve interesantnije i s velikim rastom, o čemu svjedoči značajan broj radova i nekoliko međunarodnih konferencija, koje se redovito održavaju na tu temu. Glavnina dosadašnjih istraživanja i radovi glavnih istraživača na području generativnog programiranja (Don Batory, Ulrich Eisenecker, Krzysztof Czarnecky i drugi) odnose se na primjenu principa generativnog programiranja na razvoj aplikacija u jezicima objektno-orijentiranog programiranja, kao što su C++, Smalltalk i Java. Obzirom na to da aktualni trendovi razvoja programskih jezika ukazuju da će u budućnosti jezici skripata dominirati u programiranju, a posebno na području jezika za razvoj Web aplikacija, što je detaljnije obrađeno u okviru magistarskog rada (Radošević, 2001.), potrebno je istražiti mogućnosti primjene jezika skripata za generativni razvoj aplikacija, budući da je to područje do sada malo istraženo.

U radu je predložen koncept generativnog programiranja, koji se temelji na jezicima skripata.

1.1 Ciljevi rada

Glavni cilj je kreirati novi pristup programiranju koji će se temeljiti na generativnom programiranju u jezicima skripata, što će rezultirati skraćenjem razvojnog ciklusa aplikacija, optimizacijom performansi programa i pojednostavljenjem održavanja.

U okviru tog glavnog cilja, definirani su podciljevi:

1. Na temelju istraživanja aktualnih tehnologija na području programiranja, u prvom redu jezika skripata, te postojećeg koncepta generativnog programiranja, utvrditi mogućnosti za kreiranje novog koncepta generativnog programiranja, koji će se temeljiti na oba područja. Povezivanjem tih dvaju područja postići će se nova svojstva, koja će omogućiti skraćenje razvojnog ciklusa aplikacija, optimizaciju performansi, te pojednostavljenje održavanja.

2. Kreirati novi koncept generativnog programiranja koji će biti temeljen na jezicima skripata. Definirana je njegova arhitektura i postupak generiranja aplikacija, te su istražene mogućnosti automatizacije izrade generatora kroz parametrizaciju njihovih dijelova, odnosno izrade automatiziranog sustava za izradu generatora aplikacija iz zadane klase.

3. Testirati primjenu koncepta generativnog razvoja aplikacija u jezicima skripata na izradi nekoliko generatora. Utvrđene su mogućnosti primjene tog koncepta na području Web aplikacija u jezicima skripata.

1.2 Hipoteze

Glavna hipoteza je da danas aktualni principi programiranja, generativno programiranje i jezici skripata mogu evoluirati u novi koncept generativnog programiranja za jezike skripata. Novi koncept će sadržavati naslijeđena svojstva nabrojenih principa programiranja, ali i neka nova svojstva i prednosti:

1. Izbjeći će se određene slabosti postojećeg koncepta generativnog programiranja, koji se temelji na objektno-orijentiranim programskim jezicima:

- problemi koji proizlaze iz složene sintakse i visoke razine tipizacije objektno-orijentiranih programskih jezika, koje je uočio Ousterhout (Ousterhout, 1998.,str. 5)
- budući da se jezici skripata interpretiraju, izostavit će se faza prevođenja u razvoju generatora.
- zbog cikličkog razvoja aplikacija potrebne su česte promjene u kodu generatora, što je jednostavnije izvesti kod jezika skripata, jer je taj programski kod znatno više razine od sistemskih programskih jezika, kao što su strukturni i objektno-orijentirani (uočio Ousterhout (Ousterhout, 1998.,str. 5), vidljivo iz pregleda koji je izradio Jones (Jones, 1997. str 1-16)).

2. Generator će prema novom konceptu generativnog programiranja biti potpuno odvojen od koda generirane aplikacije. To je bitna prednost zbog bolje preglednosti programskog koda za definiranje opisa aplikacije ili pojedine komponente, odnosno koda generatora.

3. U novom konceptu koriste se mogućnosti jezika skripata, kao što je Perl, u obradi znakovnih nizova kod generiranja aplikacija na temelju predložaka. To pretpostavlja izradu odgovarajućeg mehanizma u jezicima skripata, umjesto korištenja ugrađenih mogućnosti objektno-orijentiranih programskih jezika, ali se postiže veća fleksibilnost u programiranju zbog izbjegavanja sintaksnih problema i problema rukovanja složenim objektima.

1.3 Postignuti rezultati

U okviru ovog rada provedena su istraživanja sa svrhom definiranja koncepta generativnog programiranja koji se temelji na jezicima skripata, umjesto na objektno-orijentiranim programskim jezicima, kako su u početku zamislili glavni autori na području generativnog programiranja (D. Batory, K. Czarnecky, U. Eisenecker i drugi). U toku tih istraživanja u svijetu su pokrenuti i drugi projekti koji se bave sličnom tematikom, kao što je projekt razvoja jezika "Open Promol", te projekt "Codeworker". Oba ta projekta usmjerena su na razvoj novih programskih jezika, koji pripadaju jezicima skripata, specijaliziranih za izradu generatora aplikacija.
U ovom radu naglasak je stavljen na izradu grafičkog modela koji omogućuje modeliranje generatora, a ne na izradu novog programskog jezika, jer se u toku istraživanja pokazalo da se neki od postojećih jezika skripata, u prvom redu Perl, mogu iskoristiti u okviru generativnog programiranja, odnosno, već imaju odgovarajuća poželjna svojstva. Utvrđeno je da se slijedeća svojstva jezika skripata mogu iskoristiti za razvoj generatora:

- visoka razina jezika skripata, zajedno s niskim stupnjem tipizacije omogućuje jednostavnije modifikacije generatora u okviru generativnog razvoja aplikacija,

- mogućnosti jezika skripata u obradi znakovnih nizova omogućuju jednostavniju implementaciju funkcija generiranja,
- visoka fleksibilnost jezika skripata koja uključuje korištenje polja neograničene duljine te funkcija s neograničenim brojem parametara također pojednostavljuje izradu generatora,
- mogućnost samoevaluacije (izvršavanja znakovnih nizova kao programskog koda) daje sasvim nove mogućnosti u odnosu na sistemske programske jezike, koji razlikuju izvorni od izvršnog programskog koda. Samoevaluacija daje mogućnost razvoja novih arhitektura generatora koji bi generirani programski kod smještali u radnu memoriju gdje bi se oni izvršavali, bez potrebe za prevođenjem.

Postignuti su slijedeći ciljevi:

1. Definiran je koncept generativnog programiranja temeljen na jezicima skripata, i to:

- definirana je arhitektura sustava za generativno programiranje,
- definiran je skriptni model generatora, kao grafički model koji se sastoji od dva dijagrama:

  - dijagram parametara aplikacija i
  - dijagram metaskripata

- definiran je generativni razvoj aplikacija, kao postupak u kojem se paralelno izrađuju aplikacije i razvijaju generatori.

2. Razvijeno je nekoliko generatora prema skriptnom modelu. Razvijeni generatori omogućuju automatiziranu izradu web aplikacija u jezicima skripata, koje omogućuju funkcionalnosti kao što su:

  - daljinsko održavanje baza podataka,
  - korisničke web aplikacije (prikaz informacija iz baze podataka),
  - razni web upitnici.

Pokazalo se da generatori izrađeni prema skriptnom modelu mogu generirati programski kod u različitim programskim jezicima, kao što su Perl, VBscript, HTML, te C++.

3. Provedeno je testiranje razvijenih generatora obzirom na postignuta svojstva vezana uz:
  - skraćivanje razvojnog ciklusa aplikacija,
  - optimizaciju performansi generiranih aplikacija i
  - pojednostavljenje održavanja aplikacije.

Znanstveni doprinos rada sastoji se u slijedećem:

- primjena jezika skripata u okviru generativnog programiranja je do sada malo istraženo područje, jer se dosadašnja istraživanja i radovi glavnih istraživača na području generativnog programiranja (Don Batory, Ulrich Eisenecker, Krzysztof Czarnecky i drugi) odnose na primjenu principa generativnog programiranja na razvoj aplikacija u jezicima objektno-orijentiranog programiranja, kao što su Smalltalk, C++ i Java. Također, modeliranje na području jezika skripata je do sada malo istraženo područje. U okviru ovog rada definiran je koncept generativnog programiranja temeljen na jezicima skripata, te u okviru toga:

- definiran je skriptni model generatora aplikacija, kao grafički model generatora aplikacija prema predloženom konceptu generativnog programiranja, temeljenom na jezicima skripata i
- definiran je generativni razvoj aplikacija, kao postupak razvoja programa i generatora aplikacija prema predloženom konceptu generativnog programiranja.

Utoliko se ovaj rad razlikuje od navedenih projekata vezanih uz primjenu jezika skripata u okviru generativnog programiranja (Open Promol i Codeworker).

1.4 Struktura rada

Rad je podijeljen na 8 poglavlja, uključujući uvod i zaključak, te prilog. U uvodnom dijelu definirani su ciljevi rada, hipoteze, postignuti rezultati sa znanstvenim doprinosom i struktura rada.
Drugo poglavlje daje pregled aktualnih tehnologija i trendova na području programiranja, uključujući temeljne discipline ovog rada: jezike skripata i generativno programiranje.
Treće poglavlje daje pregled dostignuća dvaju dosadašnjih projekata primjene jezika skripata u okviru generativnog programiranja.
Četvrto poglavlje je centralno u okviru ovog rada jer obrađuje razvijeni skriptni model generatora koji se temelji na dva grafička dijagrama. U okviru tog poglavlja obrađeni su osnovni pojmovi novog koncepta generativnog programiranja, te je razvijen primjer jednostavnog generatora aplikacija u jeziku C++. Data je gramatika skriptnog modela, te je razvijeni skriptni model stavljen u odnos s postojećim objektnim modelom, koji se temelji na UML-u.
Peto poglavlje obrađuje generativni razvoj aplikacija kao iterativni proces u okviru kojeg se paralelno razvijaju aplikacije i generatori aplikacija. Ukazano je na prednosti takvog razvoja aplikacija, prvenstveno zbog ponovne iskoristivosti generatora i pojednostavljenja održavanja aplikacija.
Šesto poglavlje obrađuje jedan projekt primjene generativnog razvoja aplikacija. U okviru tog projekta razvijena je web aplikacija za ispitivanje zadovoljstva gostiju jednog hotela.
Sedmo poglavlje odnosi se na testiranje primjene koncepta generativnog programiranja temeljenog na jezicima skripata. Testovi se odnose na skraćivanje razvojnog ciklusa aplikacija, optimizaciju performansi generiranih aplikacija i pojednostavljenje održavanja aplikacije.

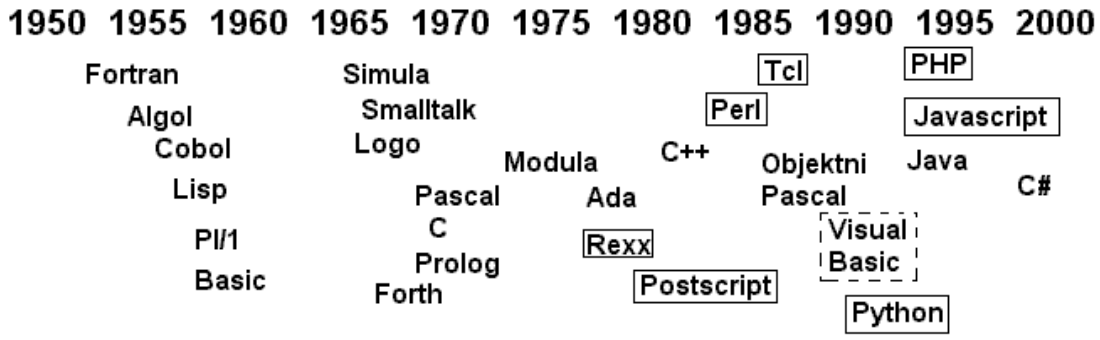
Zaključak sumira rezultate provedenih istraživanja.

U prilogu se nalazi pojmovnik generativnog programiranja prema konceptu temeljenom na jezicima skripata.

2 AKTUALNE TEHNOLOGIJE I TRENDOVI NA PODRUČJU PROGRAMIRANJA

Danas je u upotrebi niz programskih jezika i tehnika, koje bi smo, prema Ousterhoutu (1998., str 26.) mogli podijeliti u područje sistemskih programskih jezika i u jezike skripata. Osim samih programskih jezika i tehnika, razvijaju se i tehnike modeliranja, te automatskog programiranja, u koje spada i generativno programiranje.

Slika 2.1 daje pregled vremena pojavljivanja pojedinih programskih jezika, u svojim početnim inačicama. Možemo vidjeti da su jezici skripata zastupljeni pretežno među novijim programskim jezicima. Visual Basic, prema (Ousterhout, 1998.,str.27) djelomično spada u jezike skripata, a djelomično u sistemske programske jezike.



Slika 2.1.: Razvoj programskih jezika - uokvireni su jezici skripata prema (Levenez, 2004., str. 1)

2.1 Razvoj programskih jezika

U ovom dijelu rada dan je pregled razvoja programskih jezika obzirom na dvije osnovne karakteristike, to su:

- razina programskog jezika i
- stupanj tipizacije.

Razinu programskog jezika možemo izraziti dvojako :

- u broju strojnih instrukcija koje računalo mora izvršiti za jednu instrukciju zadanog višeg programskog jezika (Ousterhout,1998., str. 26.) ili
- u broju instrukcija zadanog programskog jezika koje su potrebne da bi se realizirala jedna funkcijska točka u programu (Jones, 1996., str 1.), pri čemu metriku funkcijskih točaka možemo definirati kao sredstvo za mjerenje veličine i produktivnosti softvera. Osnovnu funkcijsku točku definiramo kao jednu poslovnu funkciju krajnjeg korisnika, kao što je npr. jedan upit za ulaz podataka. Broj funkcijskih točaka u programu utvrđujemo brojanjem ulaza, izlaza, upita i logičkih grupiranja podataka sadržanih u jednoj aplikaciji i određivanjem faktora složenosti svakog od njih (Van Doren, 1997., str. 1.).

Prema tome, programski jezik je to više razine što više strojnih instrukcija je potrebno za izvršenje jedne instrukcije tog jezika, odnosno, što je manje instrukcija tog jezika potrebno za jednu funkcijsku točku.

Općenito se može reći da razina programskih jezika raste s njihovim razvojem, kao što možemo vidjeti u slijedećoj tabeli, preuzetoj od (Jones, 1996., Table 2, str. 2-14.):

Jezik	**Razina (broj strojnih naredbi po instrukciji)**	**Prosječan broj instrukcija po funkcijskoj točki**
1. generacija jezika	1	320
2. generacija jezika	3	107
3. generacija jezika	4	80
4. generacija jezika	16	20
5. generacija jezika	70	5

Tablica 2.1 : Programski jezici i razine

Prema većini istraživanja, razina programskog jezika doprinosi produktivnosti programiranja, tako Jones navodi slijedeće rezultate (Jones, 1996., str 1.):

Razina jezika (broj strojnih naredbi po instrukciji)	**Prosječna produktivnost po osobi mjesečno (u broju funkcijskih točaka)**
1 - 3	5 - 10
4 - 8	10 - 20
9 - 15	16 - 23
16 - 23	15 - 30
24 - 55	30 - 50
više od 55	40 - 100

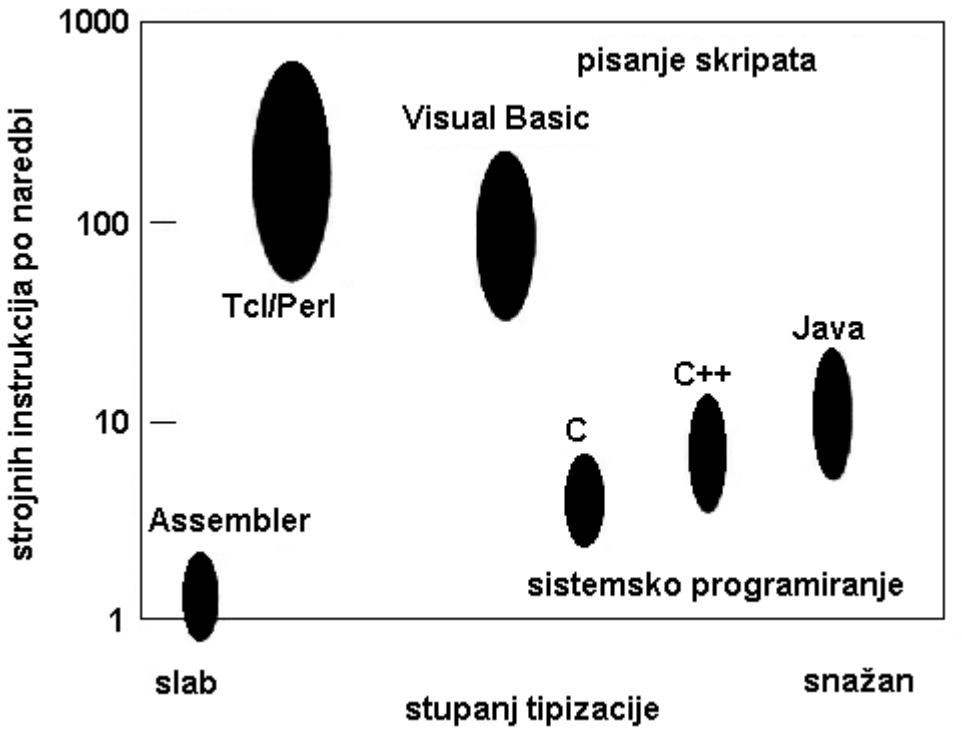
Tablica 2.2 : Odnos razine jezika prema produktivnosti programera

Usporedimo li tablice 2.1 i 2.2 možemo zaključiti da razvoj programskih jezika ide u smjeru jezika sve više razine koji donose i sve veću produktivnost programera, koji treba sve manje instrukcija za realizaciju pojedinih funkcijskih točaka programa

Tipizacija predstavlja stupanj u kojem je značenje informacije specificirano prije njenog korištenja (Ousterhout, 1998, str. 25.). U strogo tipiziranom jeziku programer definira kako će se koristiti svaki dio informacije i prevoditelj sprečava njeno korištenje na bilo koji drugi način. U slabo tipiziranom jeziku nema unaprijed zadanih restrikcija u odnosu na način korištenja informacija – značenje informacije se definira načinom njenog korištenja. Moderna računala su u osnovi bez tipa jer se svaki memorijski sadržaj može tumačiti na više načina (strojna instrukcija, cijeli broj, znak,...), za razliku od programskih jezika strukturnog i objektno-orijentiranog programiranja, na primjer (Radošević, 2001., str. 17):

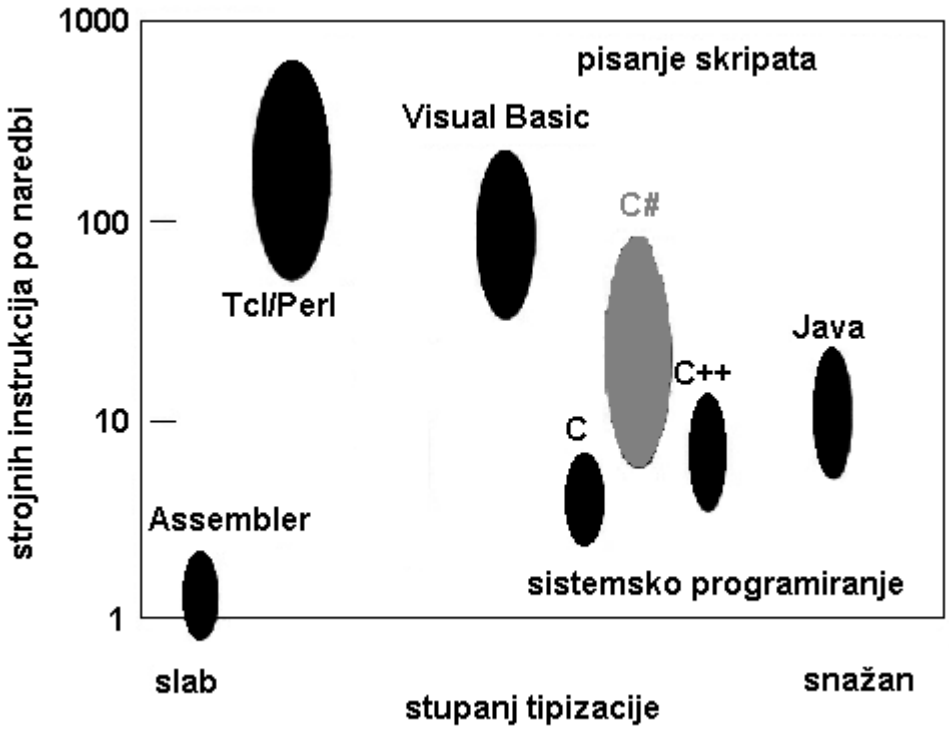
- svaka varijabla mora biti deklarirana zajedno sa svojim tipom, koji onda definira skup operatora koji se na njega mogu primijeniti, odnosno, može se koristiti isključivo na način koji je primjeren tom tipu,
- podaci i programski kod su odvojeni, što otežava česte izmjene koda,
- više varijabli može biti uključeno u strukture ili objekte s precizno definiranom podstrukturom i procedurama ili metodama za njihovu manipulaciju. Objekt jednog tipa ne može se koristiti na mjestu gdje se očekuje objekt drugog tipa.

U razvoju programskih jezika do sredine 90-tih godina prošlog stoljeća dominirao je trend sve veće tipizacije programskih jezika, što je osobito prisutno kod objektno orijentiranih programskih jezika C++ i Jave. Međutim, razvoj jezika skripata vodi s druge strane u sve manju tipizaciju, dok trend povećanja razine jezika ostaje i dalje, kao što možemo vidjeti na slici 2.2 (Ousterhout, 1998, str. 25.) :



Slika 2.2 : Usporedba različitih programskih jezika obzirom na razinu jezika i stupanj tipizacije

Pridodamo li grafikonu sa slike 2.2. programski jezik C#, koji se kasnije pojavio, dobit ćemo slijedeću sliku (Radošević,2001., str. 43.):



Slika 2.3: C# popunjava prazninu između sistemskih objektno-orijentiranih programskih jezika i jezika skripata

Na slici 2.3 možemo uočiti određeni zaokret u razvoju programskih jezika, jer se odustalo od daljnjeg povećanja stupnja tipizacije programskog jezika, ali je ostvareno povećanje razine jezika u odnosu na prijašnje sistemske programske jezike.

2.1.1 Početni programski jezici

Prvi viši programski jezici pojavili su se 50-tih godina prošlog stoljeća (slika 2.3). Ispočetka se radilo samo o simboličkom obliku strojnog jezika, do pojave Fortrana (Aaby, 1986., str 11.). Programski jezik Pascal pojavio se početkom 70-tih godina i uzet je kao prekretna točka u programiranju jer je tek u Pascalu dosljedno primijenjen koncept strukturnog programiranja, koji je dugo dominirao u programiranju, a prisutan je i danas. Prije Pascala u programiranju su dominirali programski jezici koje danas nazivamo nestrukturnim, među kojima su najviše došli do izražaja slijedeći :

- jezici niske razine, u prvom redu simbolički strojni jezici - Asembleri (engl. Assembly languages) i
- viši programski jezici, među kojima su najviše došli do izražaja COBOL za poslovne namjene i FORTRAN za matematičke proračune.

Jezici niske razine pojavili su se iz potrebe za humanizacijom programiranja : lakše je pamtiti tzv. mnemonike (riječi koje predstavljaju odgovarajuće strojne instrukcije) nego binarne brojeve koje razumije računalo. Takve jezike nazivamo simboličkim strojnim jezicima ili Asemblerima. Njihove su osnovne karakteristike slijedeće :

- jedan mnemonik u pravilu odgovara jednoj strojnoj instrukciji računala,
- skup instrukcija (mnemonika) i mogućnosti variraju u zavisnosti od procesora računala na kojem su implementirani i

- u pravilu je potreban velik broj instrukcija takvih jezika da bi se ostvarila smislena operacija, odnosno funkcijska točka.

Kasnije je uvedena mogućnost da se više mnemonika grupira, tako da čine tzv. makro instrukciju, čime se odstupilo od pravila da jedan mnemonik predstavlja jednu strojnu instrukciju računala.

Viši programski jezici uvode značajnu novinu u područje programiranja : jedna instrukcija takvog jezika više ne odgovara jednoj (ili eventualno nekoliko) strojnih instrukcija. To znači da se odgovarajuća instrukcija može realizirati na više različitih načina, pa i na međusobno posve različitim računalima. To čini programe pisane u takvim jezicima (barem teoretski) prenosivim između različitih računala.
S druge strane, došlo je do diferenciranja programskih jezika, i to najprije u dvije grupe :

- poslovni programski jezici, kao što je COBOL. Njihova osnovna karakteristika je da vrše relativno jednostavnu obradu nad velikom masom podataka i
- jezici namijenjeni matematičkim i logičkim proračunima, kao što je FORTRAN. Njihova je karakteristika da rade složenu obradu nad relativno malom količinom podataka.

Takvi, viši programski jezici uveli su u programiranje osnovne programske konstrukte koji su uključeni u kasnije strukturne programske jezike.

2.1.2 Strukturni pristup programiranju

Koncept strukturnog programiranja dugo je dominirao u programiranju, a prisutan je i danas. Njegove tri osnovne postavke su slijedeće (Motik, Šribar, 1997., str. 2.) :

- **stroga hijerarhijska struktura programa** (što znači da je upotreba skokova, u prvom redu naredbe GO TO nepoželjna, ili barem svedena na minimum),
- **odvajanje definicije podataka od njihove obrade** (odnosno, postoje odjeljci u programu za definicije podataka i odjeljci za njihovu obradu) i
- **potprogrami**. Logičke cjeline unutar programa izdvajaju se u potprograme s točno određenom zadaćom, tako da se isti programski kod može pozivati iz različitih dijelova programa. Strukturni program možemo promatrati programa kao niz jednostavnih programskih odsječaka, potprograma (Motik, Šribar, 1997., str. 3.). Svaki potprogram predstavlja program u malom : ima svoj naziv, ulazne i izlazne vrijednosti, te svoje lokalne podatke. Zamišljen je tako da predstavlja logičku cjelinu unutar programa, što olakšava kasnije ispravljanje eventualnih pogrešaka, te pozivanje iz različitih dijelova programa, tako da isti programski kod nije potrebno pisati na više mjesta u programu. Odnosno, cijeli program je presložen da bi se mogao razumjeti, pa ga se zato dijeli na više manjih dijelova – potprograma, koji se mogu realizirati pomoću naredbi odgovarajućeg programskog jezika.

Programski jezik koji najdosljednije primjenjuje koncepte strukturnog programiranja je Pascal. Pascal je nastao početkom 1970-tih godina kao rezultat analize stanja tadašnjih programskih jezika, koju je proveo Wirth, a rezultirala je Wirthovim izvještajem pod nazivom "Referentna definicija Pascala", (Wirth, 1989., str. 141-199.). U tom izvještaju Wirth se osvrnuo na stanje tadašnjih programskih jezika, kod kojih je uočio da imaju karakteristike i konstrukcije koje se ne mogu uvijek uvjerljivo logički objasniti, odnosno nisu u skladu sa sistematičnim načinom razmišljanja. Zbog

toga je odredio dva osnovna cilja, koje treba postići novim programskim jezikom , ", (Wirth, 1989., str. 141.), citat :

> - "...stvoriti jezik koji je pogodan za učenje programiranja kao sistematske discipline, zasnovane na određenim fundamentalnim konceptima koji se jasno i prirodno odražavaju u jeziku " i
> - "...razviti implementacije jezika koje su istovremeno i pouzdane i efikasne na trenutno dostupnim računalima." (kraj citata)

Pascal se do danas uglavnom zadržao kao jezik za učenje programiranja, ali važnije od toga je da je to jezik koji je postavio nove standarde u programiranju, odnosno mnogi programski jezici koji su uslijedili nakon Pascala, naslijedili su njegove osnovne konstrukte, dok im se implementacija razlikuje. Analizom osnovnih programskih konstrukata jezika Pascal i C došlo se do slijedeće tablice (Radošević, 2001., str. 22):

Programski konstrukt	**Pascal**	**C**
sekvenca	BEGIN-END	{ }
selekcija	IF, CASE	if, switch
iteracija	FOR, WHILE, REPEAT-UNTIL	for, while, do-while
potprogrami	procedure i funkcije	funkcije
slogovi	RECORD	struct
razlikovanje malih/velikih slova	NE	DA, ključne riječi obavezno malim slovima
aritmetički operatori	+ - * / div mod	+ - * / % unarni operatori
pokazivači	^	*
reference	funkcija ADDR	&
operator pridruživanja	:=	= operatori obnavljajućeg pridruživanja
relacijski operatori	= < > <> <= >=	== < > != <= >=
logički operatori	and or not	&& || !
bitovni operatori	and or not shl shr	& | ~ << >>
operator dodjele tipa	ne postoji (koriste se funkcije)	postoji
uvjetni operator	ne postoji	?
tip logičkih izraza	boolean (logički)	int (cjelobrojni)
komentar	{ i } ili (* i *)	/* i */

Tablica 2.3 : Odnos između Pascala i C-a

Tablica 2.3 pokazuje da je većina osnovnih programskih konstrukata jezika Pascal prisutna i u jeziku C, bez obzira na različitost implementacije.

2.2 Objektno orijentirani pristup programiranju

Strukturni pristup programiranju donio je značajan napredak u programiranje jer je omogućio bolju preglednost programa i omogućio ponovno korištenje dijelova programskog koda, potprograma. Potprogrami vrše zadanu obradu nad podacima koji se prosljeđuju u obliku parametara, odnosno, prema podacima koje treba obraditi, programer izabire odgovarajući potprogram. To dobro funkcionira ako su podaci koje treba obraditi uniformni, odnosno, ako mogu biti deklarirani unutar istog tipa. Što više ti podaci međusobno divergiraju, to će programer biti prisiljen upotrijebiti više selekcija u svom programu, kako bi svakom podatku pridružio potprogram koji će izvršiti potrebnu obradu (Chroboczek, 1998., str. 1-16).

Da bi se riješio taj problem, prišlo se razvoju objektno orijentiranih programskih jezika. Objektno orijentirano programiranje je tehnika za povećanje produktivnosti, kvalitete i inovacije u razvoju softvera (Guerraoui, 1996. str. 1.). Dok su procedure (potprogrami) sljedovi instrukcija koji definiraju transformaciju ulaza u izlaze, objekti su kolekcije operacija koje dijele zajedničko stanje i nude zadane usluge u bilo koje vrijeme. Proceduralno orijentirani sustav sastoji se od slijedno izvršavajućih potprograma, dok se objektno-orijentirani sustav sastoji od kolekcije objekata koji međusobno komuniciraju. Objekti predstavljaju izdvojene sustave koji nude zadane usluge (Ramos-Pollan, 2002., str. 2). Također, objekti mogu pokrenuti potrebne operacije u drugim objektima. Objekti posjeduju svoje *sučelje*, preko kojeg korisnici mogu pristupiti operacijama i atributima objekta, dok se *implementacija* skriva od korisnika (Budd, 1998., str. 3).

Objektno orijentirano programiranje donosi modeliranje, ponovno korištenje programskog koda i integraciju podataka i postupaka za njihovu obradu (Guerraoui, 1996. str. 1., Flanagan, 1997., str. 53).

Modeliranje. Objektno orijentirano programiranje prenosi strukturu problemske domene na strukturu programa, jer postoji direktan 1:1 odnos između objekata problemske domene i objekata kompjutorskog modela. Modeliranje se provodi korištenjem klasa, nasljeđivanjem i virtualnim funkcijama (Devis, 1997., str. 3.). Danas najpopularnija tehnika objektnog modeliranja je UML (eng. Unified Modelling Language).

Ponovno korištenje programskog koda i proširivost. Objektno orijentirani jezici omogućuju nasljeđivanje klasa. Nove (izvedene) klase mogu redefinirati pojedine dijelove osnovnih klasa, odnosno dodati im nove atribute, što je osobito korisno za brz razvoj prototipa. Također, postoji mogućnost definiranja tzv. *apstraktnih klasa*, čime se mogu grupirati objekti iz izvedenih klasa.

Integracijski mehanizmi. Objektno orijentirano programiranje zahvaljujući prirodi svojih mehanizama predstavlja integrativni okvir za organizaciju znanja o problemskoj domeni kroz različite faze životnog ciklusa softvera. Iste strukture koje su zadane klasama mogu se koristiti za analizu, dizajn, implementaciju i održavanje faza životnog ciklusa softvera.

Objektni pristup programiranju uvodi pojmove klase i objekta. Klasa udružuje podatke i potprograme za obradu tih podataka – metode. To udruživanje podataka i operacija nad tim podacima naziva se *enkapsulacija*. Cijeli program podijeljen je na više zatvorenih cjelina, objekata, koji međusobno surađuju u rješavanju problema. Nisu svi podaci ni sve operacije jednog objekta vidljivi izvan tog objekta, odnosno, postoji

vanjsko sučelje objekta, dok detalji implementacije nisu dostupni izvan objekta. Slijedeće što uvodi objektno orijentirano programiranje je nasljeđivanje. Nasljeđivanjem izvedena klasa preuzima podatke i metode osnovne klase, čime se osigurava ponovna iskoristivost programskog koda, ali izvedena klasa može imati i svoje vlastite podatke i metode. Naslijeđene metode nasljeđuju implementaciju iz osnovne klase, ali mogu imati i novu, vlastitu implementaciju. U tom slučaju postiže se slijedeće važno svojstvo objektno orijentiranog programiranja : *polimorfizam*. Polimorfizam omogućuje različite implementacije istoimenih metoda, a budući da su te metode pridružene podacima unutar odgovarajućih objekata, to mogu biti i njima prilagođene (Kleinoder, Golm, 1996., str. 2; Crowe, 1999, str. 1-8).

2.2.1 Trendovi razvoja objektno orijentiranog programiranja u proteklom desetljeću

Pregled stanja i razvojnih trendova na području objektno-orijentiranog programiranja sredinom 90-tih godina dao je Guerraoui, pod nazivom "Strateški pravci objektno orijentiranog programiranja" (Strategic directions in object-oriented programming; Guerraoui, 1996.). Guerraoui je identificirao četiri glavna područja istraživanja na području objektno orijentiranog programiranja: 1. integracija tehnologija, 2. softverske komponente, 3. distribuirano programiranje i 4. potraga za nasljednicima objektno-orijentirane paradigme.

Integracija tehnologija. Objektno orijentirana tehnologija mora zadovoljiti potrebe velikih razvojnih projekata po svojoj efikasnosti, fleksibilnosti, sigurnosti i standardizaciji. Sredinom 90-tih godina jedino je C++ opće prihvaćen jezik za objektno orijentirano programiranje, dok se Java u to vrijeme tek pojavila. Prihvaćenost jezika C++ dobrim dijelom je posljedica kompromisa koji su u njemu učinjeni (osim objektnog omogućuje i čisti strukturni pristup). Tada nisu bila ispunjena očekivanja prema kojima će objekti riješiti softversku krizu (Guerraoui, 1996. str. 2.).
Jedan od ciljeva razvoja objektno orijentiranog programiranja je postići njenu industrijsku snagu, a to je više pitanje povezivanja postojećih rješenja u zajednički okvir nego pronalaska novih rješenja (Guerraoui, 1996. str. 3.). Glavni zadaci (područja istraživanja), koji su bili predviđeni za slijedeću dekadu su (HankinHanne, Riis i Palsberg, 1996. str. 1-8.) : modeliranje (UML je tada bio u počecima), dizajn jezika, kompletnost i funkcionalnost, jednostavnost i čitljivost, odnosno : implementacija jezika, kompatibilnost zapisa dizajna i mogućnosti modeliranja među jezicima te prototipiranje razvojnog procesa.

Softverske komponente, predlošci i okviri. Da bi se prevladala tadašnja softverska kriza, koju je karakteriziralo to da je hardver postajao manji, brži i jeftiniji, softver je postajao veći, sporiji i skuplji za izgradnju i održavanje, jedan od načina je bio uvođenje predložaka za dizajniranje čime se koriste iskustva drugih programera kroz (Guerraoui, 1996. str. 3.):

(1) razmjenu znanja s drugim programerima o uspješnim softverskim arhitekturama,
(2) olakšavanje uključivanja novog dizajna ili stila arhitekture i
(3) korištenje provjerenih rješenja, odnosno izbjegavanje prevelikog lutanja u potrazi za rješenjima.

Predlošci se koriste za dokumentiranje softverske arhitekture i dizajna koji su se kroz vrijeme pokazali uspješnima, ali još ne proizvode ponovno iskoristiv kod.

Okviri omogućuju da se za neke zajedničke softverske komponente formiraju "polukompletni" kosturi koji se mogu prilagoditi nasljeđivanjem višestruko iskoristivih blokova. Dok su okviri tijesno povezani s pojedinom domenom (npr. korisničkim sučeljem), područje ponovne iskoristivosti je šire. Zadnjih godina su slijedeći proširivi okviri odigrali svoju ulogu u oblikovanju softverskih arhitektura : CORBA, Microsoft-ov DCOM, AWT-ovi kod Jave itd. dajući uspješna rješenja za česte probleme u softveru. To se koristi u komercijalnom softveru, dok su predlošci korišteni za postizanje ponovne iskoristivosti komunikacijskog softvera, ekspertizi razvoja i razvoju objektno orijentiranih komponenti (Guerraoui, 1996. str. 4.).

Distribuirano programiranje. Najrašireniji distribuirani objektni sustav je Web. Međutim, svojstva objektno orijentiranih jezika nije jednostavno implementirati na distribuirane sustave. Problem koji je uočio Guerraoui bio je da veliki distribuirani sustavi rastu evolutivno, a moraju osigurati : a.) da korisnici ne vide implementaciju objekata i b.) da su usluge tog objekta po svojoj funkcionalnosti jednake konkurentskim. Oni moraju osigurati odijeljenu kompilaciju (vremenski i prostorno). Nove klase se moraju moći dodati sustavu koji kontinuirano radi. Moraju se moći dodavati i novi tipovi, odnosno nove verzije, tj. mora se znati kada je neki objekt zapravo nova verzija postojećeg (Guerraoui, 1996. str. 6.).

Lokacija objekata (npr. Web server) u distribuiranom sustavu ima velikog utjecaja na performanse. Pozivanje lokalnog objekta daje bolje performanse od pozivanja udaljenog objekta za 3 reda veličine (Guerraoui, 1996. str. 6.)! To znači da distribuirani sustavi moraju imati mogućnost premještanja objekata, radi poboljšanja performansi, ali bez gubitka semantike ili naziva objekta. Također je neophodno osigurati sigurnost programskih jezika (provjera tipa i granice enkapsulacije). Bitno je da svi programski jezici za distribuirane sustave razumiju zajedničko objektno sučelje i pozivanje objekta. Glavni izazov bit će osiguranje semantičke uniformnosti, što znači da će se svi objekti morati tretirati uniformno bez obzira na njihovu lokaciju (Guerraoui, 1996. str. 7.). To znači i da ovisnost jedne komponente u sustavu o drugoj (udaljenoj) mora biti minimalna, kako greška ne bi uzrokovala pad cijelog sustava. Dva su suprotstavljena zahtjeva : robustnost sustava i semantička jednostavnost. Guerraoui je na tom području predvidio glavninu istraživanja.

Potraga za nasljednicima objektno-orijentirane paradigme. Usprkos očitim prednostima objektno orijentiranog programiranja u odnosu na ostale tehnologije programiranja i njenoj snazi, pokazalo se da paradigma ima svojih ograničenja. Osnovno obećanje objektno orijentiranog programiranja bilo je da objektno orijentirano programiranje daje programerima izuzetno veliku fleksibilnost. Međutim, pokazalo se da je taj čisti model izuzetno krhak u nekim situacijama, što dovodi do problema u svim fazama životnog ciklusa programa (Guerraoui, 1996. str. 8.).

Osnovni problem objektno orijentiranog programiranja je kruta struktura objektnog modela, odnosno izuzetno se teško ispravljaju pogreške u modeliranju. Jedan pristup rješavanju tog problema je tzv. *adaptivno programiranje* (Guerraoui, 1996. str. 8.). Adaptivno programiranje funkcionira tako da dijeli objektno orijentirani program u dva dijela : ponašanje i struktura sadržaja objekata (grafikon klasa). Poseban mehanizam jezika omogućuje dijelu *ponašanje* da se automatski prilagodi promjenama u grafikonu klasa. Time održavanje i evolucija programa postaju jednostavniji. Problemi, međutim, nastaju u odnosu klasa prema objektima (kad se promijene klase). Jedno od mogućih rješenja tog problema su tzv. jezici temeljeni na prototipovima, gdje nema razlike između klasa i objekata čime se nastoji "omekšati" razlika između promjene stanja objekta i promjene stanja njegove klase (Guerraoui, 1996. str. 8.).

Drugo rješenje moglo bi biti tzv. *Aspektno-orijentirano programiranje* (Guerraoui, 1996. str. 8.). U aspektno orijentiranom programiranju glavnina programa pisana je korištenjem najpogodnije tehnologije (npr. objektno orijentirano programiranje), a zatim se dijelovi koji nisu pogodni za rješavanje na taj način napišu korištenjem *jezika posebne namjene*. Rezultirajući skup programa u različitim programskim jezicima se kombinira pomoću posebne vrste kompilatora, zvanog *weaver*.

Danas je aspektno-orijentirano programiranje jedna od temeljnih disciplina nove discipline, koja se pojavila potkraj 90-tih godina: Generativno programiranje (eng. Generative Programming; Eisenecker, 1997., str. 2.).

2.2.2 Objektno modeliranje

Modeliranje predstavlja pojednostavljen prikaz realnosti i centralna je aktivnost u izgradnji dobrog softvera, prema (Milićev, 2001., str. 6). Modeliranjem se nastoje postići slijedeći ciljevi, prema (Milićev, 2001., str. 6):

- vizualizacija; model služi da se sustav vizualno prikaže onakvim kakav on je ili želimo da bude,
- specifikacija; modelom se definira struktura i ponašanje sustava,
- konstrukcija; model predstavlja uzor koji pokazuje kako sustav treba konstruirati i
- dokumentacija; model predstavlja dokumentaciju projektnih odluka.

Također, osnovni principi modeliranja, prema (Milićev, 2001., str. 6), bili bi slijedeći:

- izbor model koji se izrađuje ima ključan utjecaj na to kako se problem rješava i kako se oblikuje rješenje,
- svaki model može sadržavati različite nivoe detalja,
- najbolji modeli povezani su s realnim svijetom i
- nijedan model nije dovoljan sam za sebe; svaki netrivijalni sustav najbolje se opisuje malim skupom skoro nezavisnih modela.

2.2.2.1 UML

Na području objektnog modeliranja danas dominira jezik UML (eng. Unified Modelling Language). Prema (Milićev, 2001., str. 7), UML je jezik za objektno orijentirano modeliranje koji omogućuje:

- **vizualizaciju**, jer je UML vizualni, grafički jezik,
- **specifikaciju**, jer se pomoću UML-a formiraju precizni, nedvosmisleni i potpuni modeli,
- **konstrukciju**, pomoću jezika UML konstruira se softverski sustav koji se kasnije može implementirati,
- **dokumentiranje**, pomoću jezika UML mogu se dokumentirati zahtjevi, arhitektura, projekt, izvorni kod itd.

Dijagrami jezika UML imaju, prema (Milićev, 2001., str. 11) slijedeće karakteristike:

- dijagram je grafička prezentacija skupa elemenata koji predstavlja samo jedan pogled na jedan dio sustava,

- dijagram ne nosi semantiku i ne sadrži elementa modela. Dijagram samo predstavlja prikaz nekih elemenata modela,
- elementi modela su grupirani u pakete. Paket sadrži elemente, od kojih neki mogu biti i drugi, ugrađeni paketi. Tako se model organizira hijerarhijski.
- paket može sadržavati i dijagrame, koji prikazuju elemente tog, ali i drugih paketa.

Osnovni dijagrami jezika su, prema (Ambler, 2004., str. 1), slijedeći:

- Dijagram klasa (eng. class diagram)**,** prikazuje skup klasa, sučelja i suradnji i njihove međusobne relacije. Ti dijagrami su najčešći u objektno-orijentiranim sustavima. Dijagrami klasa bave se statičkim pogledom na dizajn sustava, a oni koji uključuju i aktivne klase, bave se i statičkim pogledom na procese u sustavu.

- Dijagram objekata (eng. object diagram), prikazuje skup objekata i njihove relacije u sustavu. Dijagrami objekata predstavljaju statičku snimku instanci stvari koje se nalaze u dijagramu klasa. Oni se bave statičkim pogledom na dizajn i procese sustava, kao i dijagrami klasa, ali iz perspektive pravih prototipova, tj. pravih objekata u sustavu.

- Dijagram slučajeva korištenja (eng. use case diagram) prikazuje skup slučajeva korištenja i glumaca (eng. actors – specijalan tip klase) i njihovih relacija. Dijagrami slučajeva korištenja bave se statičkim pogledom na slučajeve korištenja u sustavu i vrlo su važni kod organiziranja i modeliranja ponašanja sustava.

- Dijagram toka (eng. sequence diagram) i **dijagram suradnji** (eng. collaboration diagram) su tipovi interakcijskih dijagrama. Interakcijski dijagram prikazuje interakciju, koja se sastoji od skupa objekata i njihovih relacija, zajedno s porukama koje se mogu slati između njih. Interakcijski dijagram bavi se dinamičkim pogledom na sustav. Dijagram toka je interakcijski dijagram, koji daje naglasak na vremensku dimenziju i poredak poruka. Dijagram suradnje je interakcijski dijagram, koji daje naglasak na strukturalnu organizaciju objekata koji primaju i šalju poruke. Oba dijagrama su izomorfni, što znači da se svaki od njih može lako transformirati u drugi.

- Dijagram stanja (eng. statechart diagram) prikazuje automat stanja, koji se sastoji od stanja, prijelaza, događaja i aktivnosti i bavi se dinamičkim pogledom na sustav. Dijagrami stanja su vrlo važni u modeliranju ponašanja sučelja, klasa ili suradnji i daju naglasak na ponašanje objekta određeno slijedom događaja.

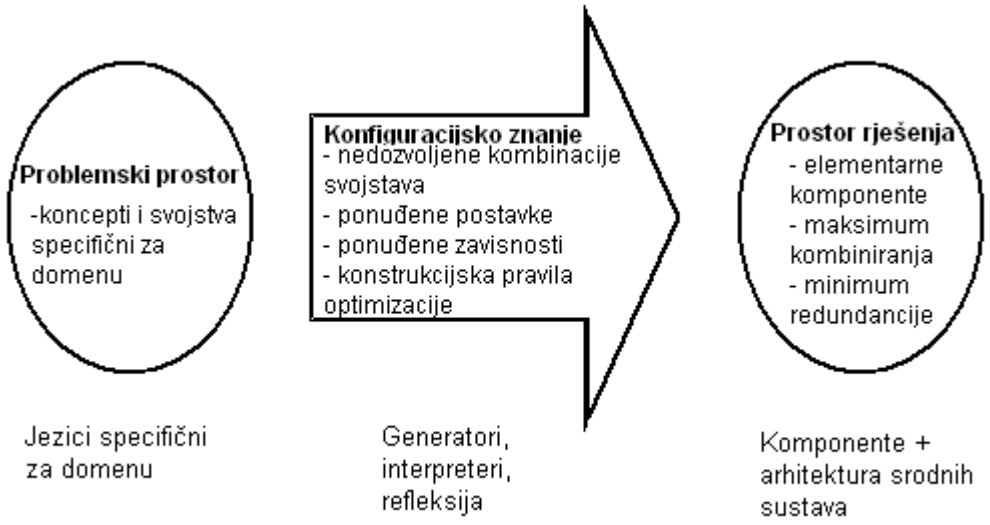
- Dijagram aktivnosti (eng. activity diagram) je specijalan tip dijagrama stanja koji prikazuje tok iz aktivnosti u aktivnost unutar sustava i bavi se dinamičkim pogledom na sustav. Vrlo su važni u modeliranju funkcija sustava i daju naglasak na tok kontrole između objekata.

- Dijagram komponenata (eng. component diagram) prikazuje organizacije i ovisnosti između skupa komponenata. Dijagrami komponenata bave se statičkim pogledom na implementaciju sustava. Povezani su sa dijagramima klasa tako da su komponente tipično mapirane na jednu ili više klasa, sučelja ili suradnji.

- **Dijagram raspoređivanja** (eng. deployment diagram) prikazuje konfiguraciju izvršnih procesnih čvorova i komponenata koje se izvode na njima. Dijagrami raspoređivanja bave se statičkim pogledom na pokretanje arhitekture sustava. Povezani su sa dijagramima komponenata tako da čvor tipično obuhvaća jednu ili više komponenata.

2.2.2.2 Generativni domenski model

Generativni domenski model (eng. Generative Domain Model - GDM) predstavlja, prema (Czarnecki, 2002, str. 1), osnovni model za automatizaciju izrade softverskih sustava. GDM se sastoji od problemskog prostora (eng. problem Space), prostora rješenja (eng. solution space) i konfiguracijskog znanja (eng. configuration knowledge), kao što možemo vidjeti na slijedećoj slici:



Slika 2.4.: Generativni domenski model (preuzeto od Czarnecki i Eisenecker, 2000, str. 38)

Za implementaciju GDM-a koriste se različite tehnologije za njegove pojedine dijelove (Czarnecki, 2002, str. 2):

- jezici specifični za domenu koriste tekstualne ili grafičke jezike (ili jezična proširenja), svojstva specifična za pojedine programske jezike ili interaktivne čarobnjake (eng. wizards),
- generatori se mogu implementirati korištenjem pretprocesora (npr. procesori predložaka), generatora aplikacija, ugrađenim mogućnostima metaprogramiranja (npr. predlošci u jeziku C++), ili proširivih programskih sustava,
- komponente se mogu implementirati korištenjem općih komponenti, kao što je standardna biblioteka predložaka (eng. Standard Template Library - STL), modela komponenata (npr. JavaBeans, ActiveX ili CORBA) ili aspektno orijentiranih programskih pristupa.

2.2.2.3 Modeliranje u okviru aspektno-orijentiranog programiranja

U okviru generativnog programiranja jedan od većih problema je modeliranje aspekata, budući da se oni ne uklapaju u osnovnu strukturu sustava nego je "presijecaju" (Kinczales et al., 1997., str. 1). Prema (Kande, 2002, str. 1) glavne pretpostavke za implementaciju aspekata unutar UML-a su slijedeće:

- da su utvrđena ona svojstva koja presijecaju osnovnu strukturu sustava (aspekti)
- da su osigurana sredstva za odvajanje presijecajućih od nepresijecajućih svojstava i smještanje ovih drugih unutar aspekata.

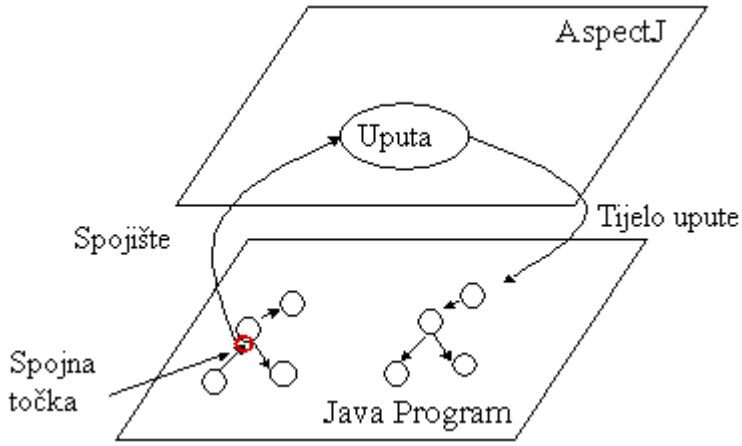
Više autora, npr. Kande (Kande, 2002, str. 1), Lieberherr (Lieberherr, 2003., str. 2) predlažu tzv. model spojnih točaka (eng. join point model) u okviru aspektno orijentiranog programiranja. Model spojnih točaka predstavlja, prema (Lieberherr, 2003., str. 2) slijedeće:

- skup svih spojnih točaka,
- mehanizam za definiranje podskupa spojnih točaka i
- mehanizam za definiranje uputa (eng. advices) na podskupovima spojnih točaka

Model spojnih točaka ime, prema (Lieberherr, 2003., str. 4), strukturu stabla. Jezik AspectJ koristi slijedeće tipove spojnih točaka, prema (Barca, 2003., str. 9):

- pozivi metoda i konstruktora,
- prihvaćanje poziva metoda i konstruktora,
- izvršenje metoda i konstruktora,
- postavljanje vrijednosti polja (eng. field set),
- učitavanje vrijednosti polja (eng. field get),
- izvršavanje potprograma za rukovanje prekidima (eng. exception handlers),
- inicijalizacija klasa i
- inicijalizacija objekata

Slika 2.5. prikazuje konceptualni model jezika AspectJ, u odnosu na temeljni jezik Java.



Slika 2.5.: Konceptualni model jezika AspectJ (preuzeto od Sung, 2002., str. 19)

Prema (Stein, Henenberg i Rainer, 2002, str.1) aspektno orijentirani model dizajna (eng. Aspect-oriented Design Model - AODM) temeljen je na UML-u i koristi njegove standardne mehanizme za proširenje kako bi omogućio izdvajanje tzv. presijecajućih pogleda (eng. crosscutting concerns). Na meta razini, AODM koristi UML-ove klasifikatore za prikazivanje spojnih točaka u obliku klasa, sučelja, čvorova ili komponenti..

Međutim, treba uočiti, prema (Prasad, 2003., str. 1) problem rukovanja tipovima (eng. typecasting), budući da spojne točke predstavljaju klase i objekte iz objektnog programiranja, što cijeli model dodatno komplicira. U ovom radu predložen je tzv. skriptni model, unutar kojega se nalazi i dijagram metaskripata, koji također predstavlja model spojnih točaka, ali su spojne točke bez tipa (eng. typeless); detaljnije o skriptnom modelu u okviru poglavlja 4.

2.3 Jezici skripata

Za sada još nema čvrste definicije jezika skripata, budući da se radi o mnoštvu jezika različitih implementacija i namjena. Ipak, većina autora slaže se u slijedećem (Radošević, 2001., str. 35) :

- **jezici skripata su vrlo visoke razine**, odnosno, na jednu instrukciju nekog od jezika skripata dolazi vrlo velik broj strojnih instrukcija računala,
- **jezici skripata su bez tipova ili slabo tipizirani** - ista varijabla može u različitim dijelovima programa sadržavati podatke različitih tipova; nema potrebe za deklaracijom podataka,
- **jezici skripata u velikoj većini slučajeva realizirani su kao interpreteri**; time se ubrzava razvoj aplikacija jer nema prevođenja, moguća je promjena programskog koda u toku izvođenja programa, nema potrebe za razvijenom razvojnom okolinom kada se žele unijeti manje promjene u programski kod i
- **jezici skripata namijenjeni su povezivanju gotovih komponenti pisanih u sistemskim programskim jezicima** (strukturnim i objektno orijentiranim).

Osim tih karakteristika, jezici skripata uglavnom teže slijedećim, zajedničkim ciljevima (Morin, Brown, 1998., str. 3):

- **neovisnost o procesoru i operativnom sustavu**; za razliku od strojnog koda, program pisan u nekom od jezika skripata može raditi u raznim ciljnim okolinama,
- **kasno povezivanje** : interni detalji vezani uz manipulaciju podacima različitih tipova rješavaju se u toku izvođenja programa, umjesto u toku prevođenja,
- **evaluacija programskog koda u toku izvođenja programa** - trebaju omogućiti generiranje ili uključivanje izvornog koda u toku izvođenja programa, a zatim taj kod evaluirati (izvršiti),
- **mnoštvo ugrađenih svojstava visoke razine** koja bi inače trebalo programirati od osnovnih programskih instrukcija (niže razine), npr. operatori za manipulaciju znakovnim nizovima, asocijativna polja itd.,
- **manje "administriranja"** - jezici skripata teže pojednostavljenju manipulacije podacima različitih tipova, alokacije memorije, korištenju ponuđenih vrijednosti varijabli itd.,
- **optimizacija za efikasnost programera** - to uključuje sposobnost za povezivanje različitih komponenti,
- **optimizaciju za određene tipove aplikacija** - npr. za Internet, obradu teksta, filtriranje podataka, administraciju sustava, grafiku, korisnička sučelja, multimediju, itd.,
- **brzo prototipiranje** - mogućnost razvoja programa u daleko kraćem vremenu nego kod sistemskih programskih jezika, kao što je C.

Ipak, preduvjet za jezike skripata je odgovarajući stupanj razvoja strojne opreme računala i operativnih sustava, naime, njihova fleksibilnost i interpretiranje dovode do značajnog pada performansi, u prvom redu brzine.

2.3.1 Slabosti objektnog pristupa

Potkraj 90-tih godina Ousterhout je u svom članku "Scripting: Higher level languages for 21th century" (Ousterhout, 1998.) dao svoju viziju daljnjeg razvoja programskih jezika, gdje je prednost dao jezicima skripata. On je tu svoju viziju dao na temelju uočenih nedostataka klasičnih tzv. sistemskih programskih jezika (strukturnih i

objektno-orijentiranih), te prednosti jezika skripata. Kao glavni nedostatak, Ousterhout navodi da objektno orijentirano programiranje nije donijelo očekivane rezultate, u prvom redu povećanje produktivnosti programera. Svojstva objektno orijentiranog programiranja kao što su stroga tipizacija i nasljeđivanje trebala bi povećati ponovnu iskoristivost softvera i riješiti mnoge probleme. Međutim, pokazalo se da jezici objektno orijentiranog programiranja povećavaju produktivnost programiranja samo za otprilike 20-30%, što je ipak daleko ispod očekivanja i daleko ispod onoga što se postiže jezicima skripata (Ousterhout, 1998., str. 29.).

Guerraoui navodi da je kruta struktura objektnog modela je glavni uzrok problema, a kao način za njihovo ublažavanje navode se dva pristupa : adaptivno programiranje i aspektno orijentirano programiranje (Guerraoui, 1996. str. 6.). Ipak, korištenje oba pristupa (adaptivno programiranje i aspektno orijentirano programiranje) ne rješava probleme objektno orijentiranog programiranja koje navodi Ousterhout (Ousterhout, 1998., str. 29.) :

- stroga tipizacija,
- objektno orijentirani programski jezici vrlo malo podižu razinu programiranja i
- orijentacija na nasljeđivanje.

Stroga tipizacija. Stroga tipizacija sistemskih programskih jezika obeshrabruje ponovno korištenje. Ona ohrabruje programere da kreiraju mnoštvo međusobno nekompatibilnih sučelja, od kojih svako zahtijeva objekte specifičnih tipova. Prevoditelj sprečava da sučelje koristi sve druge tipove objekata, čak i onda kad bi to bilo korisno. Da se to riješi, programer često mora napisati odgovarajuće potprograme za konverziju tipova (Meijer, Gough, 2002, str. 4). To zahtijeva ponovno prevođenje programa, što je danas često nemoguće, jer se aplikacije većinom distribuiraju u binarnoj formi (Ousterhout, 1998., str. 29.).

Razina programiranja. Razina programiranja, (vidi : Tablica 2.2) podiže produktivnost programiranja. Međutim, na slici 2.1 može se vidjeti da jezici objektno orijentiranog programiranja vrlo malo podižu razinu programiranja. U C++ programer i dalje radi sa sitnim jedinicama podataka koje moraju biti opisane i manipulirane jednako detaljno kao i u nekom od strukturnih programskih jezika. U principu, mogu se razviti snažne biblioteke i njihovim korištenjem podići razina programiranja. Međutim, stroga tipizacija jezika objektno orijentiranog programiranja ohrabruje samo izradu usko definiranih paketa koji su teški za ponovno korištenje (Ousterhout, 1998., str. 29.).

Orijentacija na nasljeđivanje. Drugi problem s jezicima objektno orijentiranog programiranja je njihov naglasak na nasljeđivanju. Implementacija nasljeđivanja, kod koje jedna klasa nasljeđuje programski kod koji je napisan za drugu klasu može dovesti do velikih problema, u prvom redu, otežana je manipulacija i ponovna iskoristivost. To upućuje na implementaciju klasa zajedno tako da se ni jedna klasa ne može razumjeti bez druge. Potklasa se ne može razumjeti bez znanja o tome kako su naslijeđene metode implementirane u natklasi, a natklasa se ne može razumjeti bez znanja kako su njene metode naslijeđene u potklasi. U složenoj hijerarhiji klasa ni jedna pojedina klasa ne može se razumjeti bez razumijevanja ostalih klasa u hijerarhiji. Još je nepovoljnije to što se klasa ne može izdvojiti iz svoje hijerarhije radi ponovnog korištenja. Višestruko nasljeđivanje taj problem još više pogoršava. Kao rezultat, objektno-orijentirani sustavi često pate od složenosti i manjka ponovne iskoristivosti (Ousterhout, 1998., str. 29.).

2.3.2 Primjena jezika skripata

Jezici skripata omogućuju brz razvoj aplikacija i povezivanje gotovih komponenti, dok tradicionalni, tzv. sistemski programski jezici omogućuju visoke performanse programa, u prvom redu brzinu i primjenu složenih algoritama. Radi lakšeg odlučivanja o primjeni jednih ili drugih programskih jezika, Ousterhout predlaže slijedeća pitanja (Ousterhout, 1998., str. 27.):

- Da li je glavni zadatak aplikacije povezivanje postojećih komponenti?
- Hoće li aplikacija manipulirati s mnoštvom različitih stvari (poslova, podataka)?
- Da li aplikacija uključuje grafičko korisničko sučelje?
- Sadrži li aplikacija mnoštvo operacija nad znakovnim nizovima?
- Hoće li funkcije aplikacije brzo evoluirati tokom vremena?
- Treba li aplikacija biti proširiva?

Pozitivni odgovori na ova pitanja sugeriraju upotrebu jezika skripata. S druge strane, pozitivni odgovori na slijedeća pitanja sugeriraju upotrebu sistemskih programskih jezika (strukturno ili objektno orijentiranih):

- Da li aplikacija koristi složene algoritme ili strukture podataka?
- Da li aplikacija manipulira velikim količinama podataka, npr. pikselima na slici, tako da je važna brzina izvođenja?
- Da li su funkcije aplikacije dobro definirane i sporo se mijenjaju?

2.3.2.1 Područja na kojima jezici skripata imaju široku primjenu

Zadnjih desetljeća među novim programskim jezicima pojavilo se mnoštvo jezika skripata, kao što možemo vidjeti na slici 2.1. Jezike skripata mogli bi smo, prema (Kanavin, 2003., str. 2) grupirati u slijedeće skupine:

- komandni skriptni jezici (eng. Command scripting languages)
- aplikacijski skriptni jezici (eng. Application scripting languages)
- jezici oznaka (eng. Markup languages)
- univerzalni skriptni jezici (eng. Universal scripting languages)

Komandni skriptni jezici predstavljaju najstariju klasu jezika skripata. Pojavili su se 60-tih godina prošlog stoljeća radi kontrole programa i zadača u okviru tadašnjih operacijskih sustava. Najpoznatiji takav jezik je, prema (Kanavin, 2003., str. 2) JCL (Job Control Language - jezik za kontrolu posla), koji je kreiran za operacijski sustav IBM OS/360, dok su današnji primjeri jezici ljuske (eng. shell languages), te jezici za obradu teksta, kao što su *sed* i *awk*.

Aplikacijski skriptni jezici razvijeni su 80-tih i 90-tih godina godina. Tipičan primjer je, prema (Kanavin, 2003., str. 2), Microsoftov Visual Basic, i posebno njegov podskup Visual Basic for Applications, koji je razvijen za potrebe programiranja uredskih aplikacija. Kasnije su razvijeni i VBScript, LotusScript i Javascript.

Jezici oznaka su, prema (Kanavin, 2003., str. 2), specijalan slučaj jer to nisu pravi programski jezici (s kontrolnim strukturama, tipovima i sl.), nego prije skup posebnih instrukcija koje zovemo oznakama (eng. tags). Osnovna ideja takvih jezika je odvajanje sadržaja od strukture, te također uključivanje instrukcija za formatiranje i interaktivnih objekata u dokumente (Kanavin, 2003., str. 2). Prvi takav jezik bio je

1969. IBM-om GML (Generic Markup Language; Graham, 1995., str. 21), dok su danas popularni TeX, HTML i XML.

Univerzalni skriptni jezici su danas najpoznatiji skriptni jezici. Imaju svoju primjenu u područjima u dinamičkom generiranju izvještaja (npr. Perl; Wall i Schwartz, 1991., str. 12), web stranica (npr. PHP i Perl), pristup sistemskim resursima (Python; Plosch, 1997., str. 1) i sl.. Neki od tih jezika mogu služiti i kao jezična proširenja postojećih jezika (npr. Tcl), kao njihova visoko-razinska nadgradnja (Kliček i Radošević, 2001, str. 1), ili u suradnji sa sistemskim programskim jezicima (Malinowski i Wiliamowski, 2000, str. 3), no bitno za te jezike je, prema (Ford et al, 1997., str. 4; Fedorov et al, 1998, str 17, Kanavin, 2003., str. 3) da omogućuju izradu samostalnih aplikacija, kao što su npr. web aplikacije.

Upotreba jezika skripata u okviru generativnog programiranja je relativno nova, tako da su se prvi radovi iz tog područja pojavili oko 2000. godine (npr. Štuikys i Damaševičius, 2000.).

2.3.3 Jezici bliski jezicima skripata

U novije vrijeme pojavili su se programski jezici koje djelomično možemo ubrojiti u sistemske programske jezike, a djelomično u jezike skripata. Zajednička osobina im je visoka razina programiranja, koja omogućuje brz razvoj i sintaksa koja je donekle pojednostavljena u odnosu na klasične sistemske programske jezike aplikacija (npr. Visual Basic). S druge strane, takvi jezici se ponajprije realiziraju kao prevoditelji, a ne interpreteri, što je uobičajeno za jezike skripata.

2.3.3.1 C#

C# (čita se: See Sharp) je novi Microsoftov programski jezik, dio programskog paketa Visual Studio 7, namijenjen ponajprije izradi mrežnih aplikacija za Internet. Sintaksa je u najvećem dijelu preuzeta od C++, to je objektno-orijentirani programski jezik koji nastoji ujediniti efikasnost C++ sa visokom razinom i produktivnošću Visual Basica.

Osnovne novine koje C# uvodi u odnosu na C++ su slijedeće :

- veća sigurnost rukovanja tipovima (po uzoru na Pascal) : nemoguće je varijabli dodijeliti vrijednost varijable nekog drugog tipa bez konverzije tipa (upotrebe *cast* operatora)

Primjer:

```
char c;
int i=65;
c=i;     // radi u C++, a nije dopušteno u C#
c= (char) i;
```

- rukovanje logičkim tipom podataka (bool) je također po uzoru na Pascal, odnosno, nema standardne konverzije između tipa *bool* i ostalih tipova (Hejlsberg, Wiltamuth, 2000, str. 63.). To znači da su svi logički izrazi egzaktno logičkog tipa, umjesto cjelobrojnog, kao što je to slučaj kod C i C++.

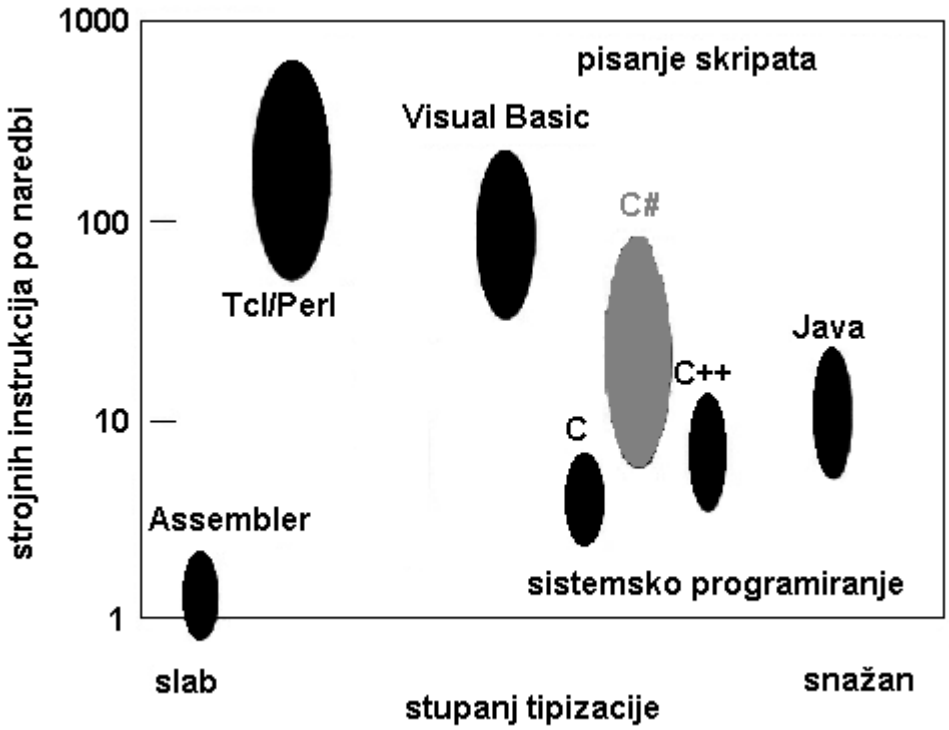
- automatska dealokacija nereferenciranih dijelova memorije (automatic garbage collection). Ova osobina bitno pridonosi produktivnosti programiranja, jer programer više ne mora svaki alocirani memorijski prostor oslobađati naredbama *free*, odnosno *delete*.

- iteracija *foreach*. Ta iteracija prisutna je u nekim jezicima skripata, kao što je Perl, a omogućuje obradu skupnih tipova podataka, kao što su vektori. Svi vektori izvedeni su iz apstraktne klase System.Array, koja definira i metode za navigaciju unutar polja, popunjavanje polja i sl.

- pojednostavljena manipulacija znakovnim nizovima (po uzoru na Pascal). Znakovnim nizovima moguće je pridruživati vrijednosti pomoću operatora pridruživanja vrijednosti (=), odnosno uspoređivati ih pomoću relacijskih operatora, umjesto korištenja funkcija iz biblioteke *string.h* .

- delegati. Delegati u izvjesnom smislu predstavljaju zamjenu za pokazivače, omogućujući međutim, veću sigurnost programskog koda, budući da pokazivači omogućuju i dohvaćanje dijelova memorije koje ne alocira program. Dobro svojstvo delegata je da može referencirati objekt iz bilo koje klase.

Na temelju gore navedenog, može se zaključiti da jezik C# predstavlja iskorak u smislu veće sigurnosti programskog koda, bolje kontrole i pojednostavljenja programiranja. U tom smislu uvedena su neka svojstva Pascala. Sve to rezultira visokom razinom jezika i većom fleksibilnošću u odnosu na C++, što C# približava jezicima skripata.
U izvjesnom smislu, jezik C# predstavlja i "kariku koja nedostaje" u razvoju programskih jezika, približavajući objektno-orijentirano programiranje jezicima skripata, tako da bi smo ga među programskim jezicima sa slike 2.1 (Ousterhout, 1998., str. 25) mogli pozicionirati kao na slijedećoj slici :



Slika 2.6 : C# popunjava prazninu između sistemskih objektno-orijentiranih programskih jezika i jezika skripata

2.3.4 Slabosti jezika skripata

Jezici skripata donose visoku razinu i fleksibilnost u programiranje, što je pogodno za brz razvoj aplikacija (Ousterhout, 1999., str. 4.). Slabosti jezika skripata očituju se u slijedećem :

a.) neefikasnost programskog koda, odnosno, programski kod jezika skripata u pravilu se sporije izvršava od programskog koda sistemskih programskih jezika, najviše zbog toga jer se jezici skripata interpretiraju.
b.) nisu pogodni za programiranje složenih algoritama - zbog svoje orijentacije na korištenje gotovih komponenti teško je pisati programe koji su unutar sebe složeni i zahtijevaju rukovanje elementarnim podatkovnim jedinicama (Hermann, Gaspary, Almeida, 1998. str. 2.; McGrath, 1998, str. 143-144.).
c.) budući da se interpretiraju, mogu se izvršavati samo unutar odgovarajućih programskih okruženja (tamo gdje postoji odgovarajući interpreter).

Razvoj strojne opreme računala i sve veća popularnost Interneta i drugih klijentsko-serverskih sustava dovodi do toga da slabosti pod (a) i (c) sve manje dolaze do izražaja, dok ostaje i dalje prigovor da jezici skripata nisu pogodni za programiranje složenih algoritama. To će, najvjerojatnije i dalje ostati područje sistemskih programskih jezika (Ousterhout, 1998., str. 28).

Također, možemo primijetiti da u okviru jezika skripata ne postoji široko prihvaćena metoda modeliranja, kao što je to UML kod objektno orijentiranog programiranja. Umjesto toga, često je shvaćanje kako za jezike skripata modeliranje, barem ono temeljeno na grafičkim dijagramima, nije potrebno, zbog visoke razine tih jezika, npr. Bezroukov (Bezroukov, 2003, str 5.). Nedostatak modeliranja mogao bi, međutim, biti ozbiljan nedostatak u okviru generativnog programiranja, gdje se danas provode

opsežna istraživanja (vidi: poglavlje 2.2.2.1: Primjena tehnika objektnog modeliranja u okviru generativnog programiranja).

2.3.5 Očekivanja od primjene jezika skripata u okviru generativnog programiranja

Upotreba jezika skripata u okviru generativnog programiranja je relativno nova, tako da se se prvi radovi pojavili oko godine 2000 (npr. Štuikys i Damaševičius, 2000.). Ipak, prema dosadašnjim rezultatima, uključujući i istraživanja u okviru ovog rada, mogli bismo očekivanja od primjene jezika skripata u okviru generativnog programiranja podijeliti u dvije skupine:

a.) prevladavanje nedostataka jezika i alata objektno-orijentiranog programiranja u okviru generativnog programiranja i
b.) korištenje dobrih svojstava jezika skripata koja se mogu iskoristiti u okviru generativnog programiranja.

Ousterhout (Ousterhout, 1998., str. 29.) je uočio daje jedna da visoki stupanj tipizacije sistemskih programskih jezika (u prvom redu objektno-orijentiranih) dovodi do krutosti objektnog modela. Aspektno orijentirano programiranje nastalo je, prema (Guerraoui, 1996. str. 6.) s ciljem da se ublaži kruta struktura objektnog modela. Ostaje, međutim problem povezivanja presijecajućih pogleda (eng. crosscutting concerns), pomoću spojnih točaka (eng. join points), gdje se, prema (Prasad, 2003., str. 1; Vranić i Navrat, 2000., str. 5), javlja problem rukovanja tipovima (eng. typecasting). Tu bi se moglo očekivati upotrebu jezika skripata, jer je jedna od glavnih namjena tih jezika, prema (Ousterhout, 1998., str. 29.), povezivanje komponenti pisanih u sistemskim programskim jezicima, zahvaljujući tome što su ti jezici bez tipa (eng. typeless), odnosno s niskim stupnjem tipizacije, čime se izbjegava problem rukovanja tipovima. S tom svrhom razvijeni su i neki specijalizirani jezici skripata, kao što je DataScript (Back, 2002., str. 66).

Također, određene prednosti jezika skripata, mogle bi se iskoristiti za realizaciju generatora aplikacija:

- mogućnost evaluacije znakovnih nizova (to je kod sistemskih programskih jezika teško izvedivo, zbog potrebe za prevoditeljem),
- mogućnosti pojedinih jezika skripata, kao što je Perl u obradi znakovnih nizova, što omogućuje jednostavne modifikacije programa, prema ()
- mogućnost korištenja polja neograničene veličine moglo bi pojednostaviti pisanje generatora jer programeru otpada problem oko alokacije memorijskog prostora i
- mogućnost korištenja potprograma s neograničenim brojem parametara, što može biti korisno za realizaciju funkcija generiranja (detaljnije u poglavlju 4).

U okviru ovog rada ponuđen je tzv. skriptni model koji treba omogućiti modeliranje generatora aplikacija temeljenih na jezicima skripata (detaljnije u poglavlju 4).

2.4 Generativno programiranje

Generativno programiranje (engl. Generative Programming) predstavlja disciplinu automatskog programiranja koja se, pod tim nazivom pojavljuje potkraj 90-tih godina 20 stoljeća. Prema definiciji, generativno programiranje predstavlja "...dizajniranje i

implementaciju programskih modula koji se mogu kombinirati radi generiranja visoko specijaliziranih i **optimiziranih** sustava koji omogućuju rješavanje specifičnih zadataka." (Eisenecker, 1997. str 2). Težnja k optimizaciji programskog koda predstavlja, prema (Eisenecker, 1997. str 2) glavnu razliku u odnosu na druge tehnike automatskog programiranja.

Osnovni ciljevi generativnog programiranja, su, prema (Czarnecki, 1999., str. 1) slijedeći:

a.) smanjenje konceptualnog razdora između programskog koda i domenskih koncepata (poznatog kao postizanje visoke intuitivnosti)
b.) postizanje visoke ponovne iskoristivosti koda i njegove prilagodljivosti
c.) pojednostavljenje upravljanja različitim varijantama komponenti i
d.) povećanje efikasnosti programskog koda (u brzini i korištenju resursa)

Da bi se to postiglo, GP postavlja slijedeće principe (Czarnecki, 1999., str. 1):

- **Parametrizacija razlika**. Parametrizacija omogućuje predstavljanje klasa komponenti, umjesto pojedinačnih komponenti.
- **Analiza i modeliranje zavisnosti i interakcija**. Nisu sve kombinacije vrijednosti parametara ispravne, tako da vrijednosti pojedinih parametara mogu uvjetovati vrijednosti nekih drugih parametara. Te zavisnosti koje se pojavljuju na istoj razini apstrakcije nazivaju se *horizontalno konfiguracijsko znanje* (engl. *horizontal configuration knowledge*).
- **Odvajanje područja problema od područja rješenja**. Problemski prostor sastoji se od apstrakcija specifičnih za pojedinu domenu s kojom rade programeri, dok se prostor rješenja sastoji od implementacijskih komponenti, npr. generičkih komponenti. Ova dva prostora imaju različite strukture i povezuju se pomoću tzv. *vertikalnog konfiguracijskog znanja (engl. vertical configuration knowledge)*, budući da se radi o interakciji parametara na različitim razinama apstrakcije. Automatska konfiguracija koristi oba (horizontalno i vertikalno) konfiguracijska znanja.
- **Uklanjanje zališnosti (redundancije) programskog koda i optimizacija prema specifičnostima domene**. Kod uobičajenog statičkog generiranja komponenti - pomoću programa prevoditelja, glavnina zališnosti odnosi se na programski kod koji se ne koristi, razne provjere i nepotrebne indirekcije, koje se mogu ukloniti.

Postojeći koncept generativnog programiranja temelji se na tehnikama i programskim jezicima objektno-orijentiranog programiranja.
Današnji jezici objektno-orijentiranog programiranja izvedeni su kao dvorazinski jezici, tako da postoje pretprocesor i prevoditelj. Pretprocesori pojedinih objektno-orijentiranih jezika, kao što je C++, imaju mogućnosti generiranja različitih instance klasa, odnosno slobodnih funkcija (npr. u C++) na temelju odgovarajućih predložaka (Mosses, 2002, str. 5), kao u slijedećem primjeru:

Primjer (C++)

```
template <class TIP>
void ispis (TIP param){
        cout << param << endl;
};

void main(){
        ispis (5);
        ispis ("Red teksta");
}
```

U primjeru se interno generiraju dvije funkcije za ispis, koje obrađuju parametre različitih tipova (ovdje cjelobrojnog tipa, odnosno znakovnog niza). Generiranjem različitih instanci klasa, za svaki pojedini tip koji klasa koristi, uklanja se potreba za interpretacijom različitih tipova u toku izvođenja programa, što doprinosi optimizaciji performansi programa. Predlošci se smještaju u tzv. aktivne biblioteke, kako bi se mogli koristiti u razvoju različitih programa, odnosno pojedinih njihovih komponenti.

Postojeći koncept generativnog programiranja razlikuje se od ostalih tehnika automatskog programiranja po tome što omogućuje različite razine automatizacije, a ne samo najvišu, razinu aplikacije kao cjeline. Generativno programiranje usmjereno je, prema (Czarnecki i Eisenecker, 2000., str. 15), u prvom redu na razvoj komponenti programa i razvoj generatora usporedno s razvojem aplikacija.

Prednosti generativnog programiranja su, prema (Czarnecki i Eisenecker, 2000. , str. 15), korištenje ugrađenih mehanizama pojedinih objektno-orijentiranih programskih jezika koje se odnose na pretprocesor, kao što su predlošci funkcija i klasa u C++, optimizacija izvornog koda programa i poboljšanje performansi programa, mogućnost korištenja tehnika objektnog programiranja, kao što su nasljeđivanje i polimorfizam, te, prema (Eichhelberger i Wolff, 2000., str. 4), primjena tehnika objektnog modeliranja, kao što je UML, za modeliranje aktivnih biblioteka.

2.4.1 Temeljne discipline Generativnog programiranja

Generativno programiranje proizašlo je iz težnje za povećanjem produktivnosti izrade softvera prelaskom na njegovu proizvodnju na način koji se može usporediti s industrijskom proizvodnjom (Czarnecki, 2002., str. 3; Ruhl, 2002., str. 4; Sells, 2002., str. 2). Zbog određenih svojih slabosti, koje je uočio Guerraoui (Guerraoui, 1996. str. 6.), odnosno Ousterhout (Ousterhout, 1998., str. 29.), objektno-orijentirano programiranje nije u mogućnosti zadovoljiti te težnje u potpunosti. Zbog toga su sredinom 90-tih godina predložene neke nove discipline u okviru programiranja, koje bi trebale naslijediti objektnu paradigmu, među kojima je i tzv. Aspektno orijentirano programiranje (Guerraoui, 1996. str. 6.), koje je danas jedna od temeljnih disciplina generativnog programiranja.

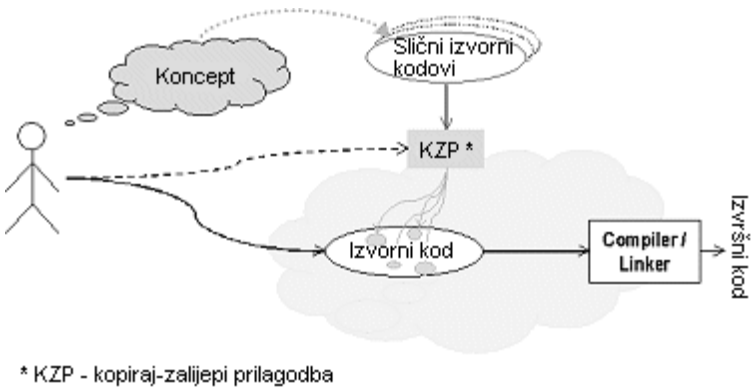
Generativno programiranje temelji se, prema (Czarnecki, 1999., str. 1), na slijedećim disciplinama:

- metaprogramiranje
- generičko programiranje
- objektno orijentirano programiranje
- aspektno orijentirano programiranje i
- domenski inženjering (engl. Domain Engineering)

2.4.1.1 Metaprogramiranje

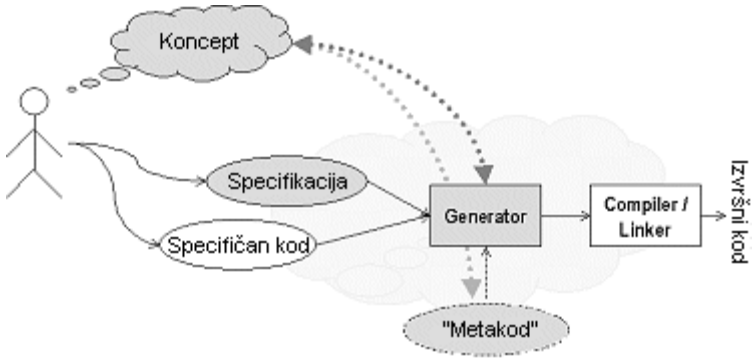
Metaprogramiranje predstavlja, prema (Cordy i Shukla, 1992., str. 2), proces definiranja predložaka koda (metaprograma) na temelju kojih se automatskim postupkom mogu izrađivati softverske komponente iz zadane klase, odnosno, prema (Stone, 1998., str. 1; Rideau, 1999., str. 1; Rodger, 2002., str. 1) metaprogramiranje možemo definirati kao izradu programa koji generiraju druge programe. Neki programski jezici, uključujući i C++ imaju ugrađene mehanizme za metaprogramiranje temeljeno na predlošcima (eng. templates). Prema (Attardy i Cisternino, 2001., str. 1; Smaragdakis, Batory, 1997. str. 4) C++ podržava generičko programiranje kroz mehanizam predložaka, što omogućuje definiranje parametriziranih klasa i funkcija. Predlošci zajedno s ostalim svojstvima tvore kompletan podjezik u okviru jezika C++, odnosno, prema (Czarnecki i Eisenecker, 2000. , str. 27) C++ možemo promatrati kao dvorazinski jezik, jer može sadržavati statički programski kod i dinamički kod, koji se izvršava za vrijeme izvođenja programa.

Osnovna ideja metaprogramiranja proizlazi iz široko prihvaćene programerske metode, koja se obično naziva kopiraj-zalijepi prilagodba (eng. Copy Paste Adapt, CPA), kao što se može vidjeti na slijedećoj slici (Delta Software, 2004., str 1.):



Slika 2.7: Kopiraj-zalijepi prilagodba

Kod kopiraj-zalijepi prilagodbe programer na temelju vlastite ideje izrađuje nove programe koristeći pritom dijelove programskog koda postojećih sličnih programa, uz potrebne modifikacije. Generatori programskog koda, prema (Delta Software, 2004., str 1.), polaze od pretpostavke da se taj ručni postupak može automatizirati. Umjesto baze gotovih programskih rješenja, generatori koriste bazu tzv. metaprograma, kao što možemo vidjeti na slijedećoj slici (Delta Software, 2004., str 1.):



Slika 2.8: Princip generatora

Na slici se može vidjeti da generatori koriste specifikaciju programa, koja zapravo predstavlja formalizirani oblik programerove zamisli, specifičan programski kod (dio programskog koda koji se ne može generirati) i tzv. "Metakod" - nedovršene predloške programa (Delta Software, 2004., str 1.).

2.4.1.1.1 Ilustrativni primjer

U slijedećem primjeru jednostavan generator generira na temelju specifikacije programski kod u C++:

Specifikacija programa:

Specifikacija programa definira oznake koje se koriste u predlošku (ovdje je to vrijednost lijevo od znaka ':') i njihove vrijednosti, s kojima se te oznake zamjenjuju (vrijednost desno od znaka ':').

prva:cjelobrojna
tip_prve:int
druga:realna

```
tip_druge:float
kod1: cout << "Zbroj ove dvije varijable iznosi " << #prva# + #druga# << endl;
```

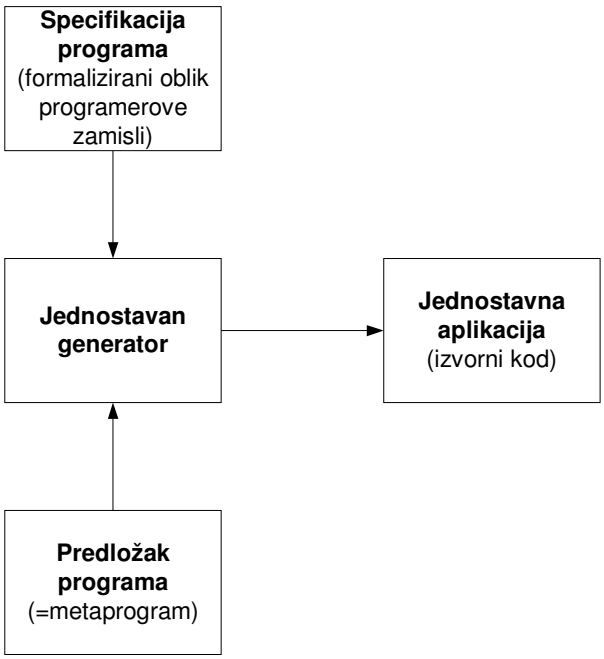
U ovoj specifikaciji programa definirane su varijable koje program koristi i specifičan programski kod koji generator uključuje u program. Treba uočiti da ovdje nije definirana struktura programa ni proces obrade (osim u dijelu koji se odnosi na specifičan programski kod), pa ni funkcionalnosti, nego samo specifične karakteristike (koje u daljnjem tekstu nazivamo aspektima) pojedinog programa u odnosu na ostale koje generator može generirati (=problemska domena generatora). Također, unutar specifičnog programskog koda korištene su oznake '#prva#' i '#druga#' koje se zamjenjuju odgovarajućim varijablama, definiranim unutar iste specifikacije. Takve oznake koriste se u metaprogramima.

Predložak programa (=metaprogram)

```
#include <iostream.h>
#tip_prve# #prva#;
#tip_druge# #druga#;

void main(){
        //unos vrijednosti varijabli
        cout << "#prva# = ";
        cin >> #prva#;
        cout << "#druga# = ";
        cin >> #druga#;
        // ispis vrijednosti varijabli
        cout << "--------------------" << endl;
        cout << "#prva# (#tip_prve#) = ";
        cout << #prva# << endl;
        cout << "#druga# (#tip_druge#) = ";
        cout << #druga# << endl;
        // specifičan programski kod
        #kod1#
}
```

Predložak programa definira strukturu, funkcionalnosti i proces obrade unutar generiranog programa. Oznake između znakova '#' odnose se na varijabilne dijelove programskog koda, koji se definira unutar specifikacije programa.
U ovom primjeru postupak generiranja možemo prikazati slijedećim dijagramom:



Slika 2.9. Postupak generiranja aplikacije kod jednostavnog generatora

Kao rezultat djelovanja jednostavnog generatora, u primjeru se dobije slijedeći programski kod u C++:

```
#include <iostream.h>
int cjelobrojna;
float realna;

void main(){
        //unos vrijednosti varijabli
        cout << "cjelobrojna = ";
        cin >> cjelobrojna;
        cout << "realna = ";
        cin >> realna;
        // ispis vrijednosti varijabli
        cout << "--------------------" << endl;
        cout << "cjelobrojna (int) = ";
        cout << cjelobrojna << endl;
        cout << "realna (float) = ";
        cout << realna << endl;
        // specifičan programski kod
        cout << "Zbroj ove dvije varijable iznosi " << cjelobrojna + realna << endl;
}
```

Ovdje vidimo da je jednostavan generator zamijenio oznake u predlošku programskog koda s vrijednostima definiranim u specifikaciji programa. Specifičan programski kod (oznaka 'kod1' u specifikaciji) također sadrži oznake ('#prva#' i '#druga#'), odnosno, predstavlja metaprogram.

U primjeru je, unutar specifikacije programa, korišten specifičan programski koda u ciljnom programskom jeziku. U slučaju da se izbjegne upotreba specifičnog koda, možemo vidjeti da specifikacija programa više ne sadrži ni jedan element specifičan za ciljni programski jezik, čime se otvara mogućnost generiranja programa u različitim programskim jezicima na temelju iste specifikacije.

2.4.1.2 Generičko programiranje

Generičko programiranje predstavlja, prema (Gibbons i Jeuring, 2003., str. 1), izradu programa takvih da budu što je moguće više prilagodljivi, a to se postiže poopćavanjem, tj. izradom generičkih programa. U dosadašnjem razvoju programiranja općenitost se nastojala postići korištenjem potprograma kod strukturnog programiranja, te klasa kod objektnog, no, generički programi u modernom smislu uključuju netradicionalne oblike polimorfizma; kao što su korištenje parametara vrlo bogate strukture, kao što su drugi programi, tipovi, klase, pa čak i cijele programske paradigme (Gibbons i Jeuring, 2003., str. 1).

Premda je generičko programiranje jedna od temeljnih disciplina generativnog programiranja, svejedno postoji bitna konceptualna razlika (Eisenecker, 1998., str. 1) između te dvije discipline, koja proizlazi iz značenja odgovarajućih riječi, na temelju kojih su nastali ti nazivi, prema (The American Heritage Dictionary of the English Language; Dictionary.com, 2004.):

- **generički** (pridjev; eng. generic): koji se odnosi ili opisuje cijelu grupu ili klasu, opći. Biol. koji proizlazi iz gena ili se odnosi na gene.

- **generativan** (pridjev; eng. generative): koji ima sposobnost izrade, proizvodnje ili stvaranja.

Prema tome, generičko programiranje bavi se, prema (Eisenecker, 1997., str. 1), izradom što općenitijih parametriziranih komponenti, koje se mogu uključivati u aplikacije. S druge strane, generativno programiranje, u odnosu na generičko, omogućuje specijalizaciju, iz koje proizlazi optimizacija programskog koda i performansi (Eisenecker, 1997., str. 1). Te dvije discipline se međutim, međusobno dopunjuju i prožimaju, prema (Eisenecker, 1998., str. 2) jer nije moguće ili nije prikladno sve komponente generirati; neke je bolje koristiti kao generičke.

2.4.1.3 Objektno orijentirano programiranje

Generativno programiranje do sada se uglavnom temeljilo na tehnikama i alatima objektno-orijentiranog programiranja. Početna istraživanja krajem 90-tih godina prošlog stoljeća, radovi glavnih istraživača na području generativnog programiranja (Don Batory, Ulrich Eisenecker, Krzysztof Czarnecky), kao i glavnina današnjih istraživanja, odnose se na primjenu principa generativnog programiranja na razvoj aplikacija u jezicima objektno-orijentiranog programiranja, kao što su Smalltalk, C++ i Java. Tek nešto kasnije, od 2000. godine pojavljuju su se prvi značajniji radovi na području primjene jezika skripata u okviru generativnog programiranja (npr. Štuikys i Damaševičius, 2000.).

Kao najvažnije razloge za primjenu tehnika i alata objektno-orijentiranog programiranja u okviru generativnog programiranja mogli bi smo izdvojiti slijedeće:

- **primjena osnovnih principa objektno-orijentiranog programiranja.** Enkapsulacija, nasljeđivanje, skrivanje podataka i polimorfizam - prema Motik i Šribar, 1997., str. 3.) imaju svoju primjenu u okviru temeljnih disciplina generativnog programiranja, u prvom redu generičkog programiranja. Generičko programiranje proširuje postojeće principe, uvodeći i netradicionalne oblike polimorfizma, npr, korištenje parametara vrlo bogate strukture, prema (Gibbons i Jeuring, 2003., str. 1)

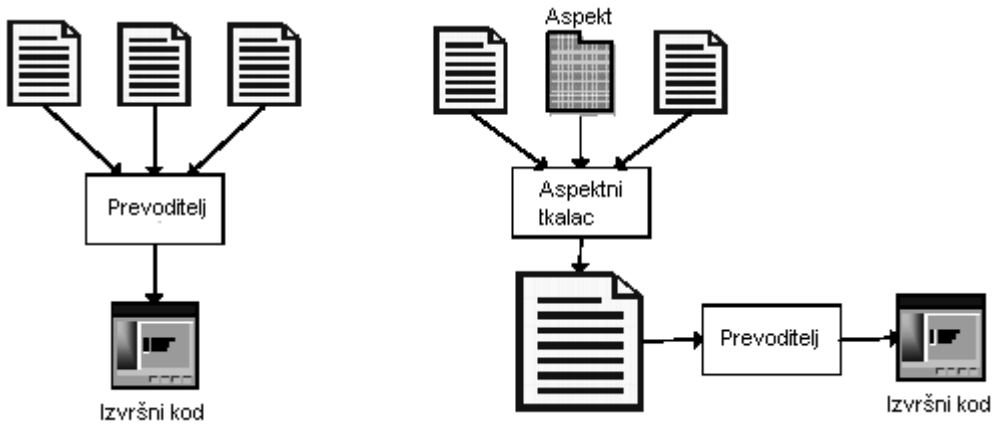
- **postojanje razvijenih tehnika objektnog modeliranja, u prvom redu tehnike UML.** U okviru generativnog programiranja danas su vrlo intenzivna istraživanja na području primjene tehnike UML za modeliranje aktivnih biblioteka, koje prema (Czarnecki, Eisenecker, Glück, Vandevoorde i Veldhuizen, 1998., str. 1) predstavljaju baze znanja za generiranje aplikacija. Međutim, na tom području postoje i problemi, prvenstveno s modeliranjem aspekata (Kuhl, 2000, str. 1,)

- **korištenje ugrađenih svojstava pojedinih jezika objektnog programiranja**, u prvom redu jezika C++ i Java. Današnji jezici objektno-orijentiranog programiranja izvedeni su kao dvorazinski jezici, tako da postoje pretprocesor i prevoditelj (Volter, Gartner, 2003., str. 2). Pretprocesori pojedinih objektno-orijentiranih jezika, kao što je C++, imaju mogućnosti generiranja različitih instanci klasa, odnosno slobodnih funkcija (npr. u C++) na temelju odgovarajućih predložaka (eng. template), što, prema (Czarnecki i Eisenecker, 2000. , str. 27) omogućuje metaprogramiranje.

- **postojeće slabosti jezika skripata**. U prvom redu nepostojanje šire prihvaćenih tehnika modeliranja.

2.4.1.4 Aspektno orijentirano programiranje

Aspektno orijentirano programiranje (eng. Aspect Oriented Programming) nastalo je, kao novi pristup u programiranju, u okviru objektnog programiranja s ciljem da se ublaži kruta struktura objektnog modela (Guerraoui, 1996. str. 6.). Možemo ga, prema (Lee K.W.K, 2002., str. 4) definirati kao tehnologiju koja omogućuje odvajanje presijecajućih pogleda (eng. crossccutting concerns) u pojedine jedinice, koje nazivamo aspektima. Aspekti prema (Kinczales et al., 1997., str. 1) predstavljaju svojstva koja "presijecaju osnovnu funkcionalnost sustava" (Kinczales et al., 1997., str. 1), odnosno svojstva koja nisu vezana za pojedine organizacijske jedinice programa kao što su funkcije ili klase, nego se mogu pojavljivati u različitim dijelovima aplikacija, prema (Lee K.W.K, 2002., str. 1; Sung, 2002, str. 1). Neki od primjera takvih svojstava, koja se ne mogu jednoznačno pridružiti pojedinim organizacijskim jedinicama programa odnose se, prema (Guerraoui, 1996. str. 8.) na sigurnost, interakciju među objektima i sinkronizaciju.



Slika 2.10.: Tradicionalni (lijevo) i aspektno orijentirani (desno) pristup izradi aplikacija; preuzeto od (Lee K.W.K, 2002., str. 4)

Važnu ulogu u aspektno orijentiranom pristupu ima tzv. aspektni tkalac (eng. Aspect Weaver), program koji uključuje pojedine aspekte u program, omogućujući mu time nova svojstva, prema (Lee K.W.K, 2002., str. 4; Gray, Lin i Zhang, 2003., str. 1).

U okviru predloženog koncepta generativnog programiranja temeljenog na jezicima skripata implementacija aspekata neophodna je radi specifikacije programa, jer se u specifikaciji zadaju samo specifični aspekti aplikacije, oni prema kojima se pojedina aplikacija razlikuje od ostalih unutar svoje problemske domene. Ulogu aspektnog tkalca u predloženom konceptu preuzima generator.

2.4.1.5 Domenski inžinjering

Domenski inženjering (engl. Domain Engineering) predstavlja, prema (Czarnecki, 1999., str. 3), metodu za kompletnu analizu i dizajn ponovo iskoristivih biblioteka u području postupaka kao što su obrada slika, numerička obrada, spremnici (eng. containers) i sl. Također, domenski inžinjering, prema (SEI, 2002., str. 1), podržava programski inženjering (eng. application engineering) koji koristi modele i arhitekture za izgradnju sustava, s naglaskom na ponovnu iskoristivost i proizvodne trake (eng. product lines). Softverska proizvodna traka predstavlja, prema (SEI, 2002., str. 2; Griss, 2000., str. 2), skup softverskih sustava koji dijele zajednički, upravljani skup svojstava s ciljem zadovoljavanja specifičnih potreba zadanog segmenta tržišta, a domenski modeli predstavljaju skup zahtjeva zajedničkih za sustave unutar proizvodne trake (Batory, 2002, str. 3).

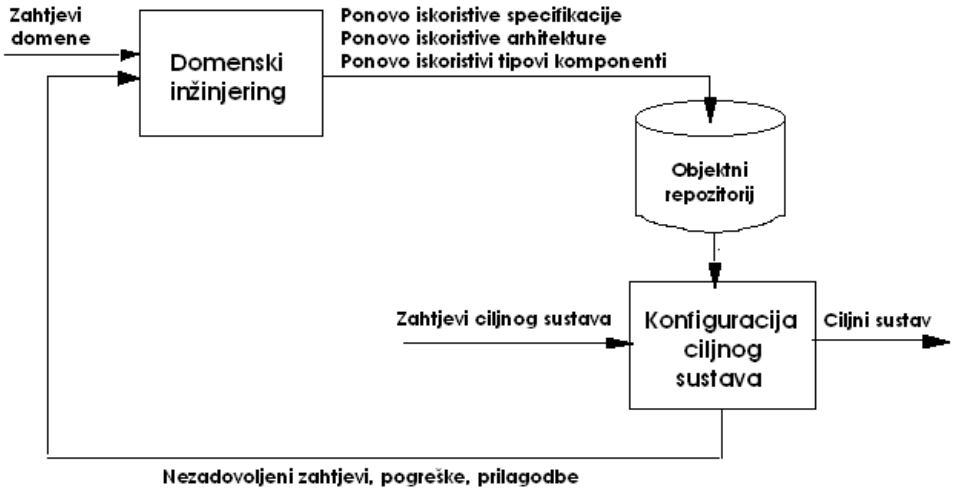
U okviru generativnog programiranja Czarnecki je, prema (Czarnecki, 1999., str. 3), predložio metodu DEMRAL (eng. Domain Engineering Method for Reusable Algorithmic Libraries" - domenska inženjerska metoda za ponovno iskoristive biblioteke algoritama). Metoda DEMRAL, prema (Czarnecki i Eisenecker, 2000, str. 283) dijeli dizajn domene na tri aktivnosti:

1. Identifikaciju i specifikaciju cjelokupne implementacijske arhitekture
2. Identifikaciju i specifikaciju jezika specifičnih za domenu (eng. domain-specific languages - DSL)
3. Specifikaciju konfiguracijskog znanja (eng. configuration knowledge)

Zadaci u okviru tih aktivnosti bili bi, prema (Czarnecki i Eisenecker, 2000, str. 283):

- određivanje opsega domenskog modela
- identifikacija programskih paketa
- razvoj arhitektura i identifikacija implementacijskih komponenti
- identifikacija korisničkih jezika specifičnih za domenu
- identifikacija interakcija među jezicima specifičnim za domenu
- specifikacija jezika specifičnih za domenu i njihovog prijevoda

Kao što se može vidjeti na slici 2.11, primjena domenskog inženjeringa treba rezultirati ponovo iskoristivim specifikacijama, arhitekturama i tipovima komponenti, od kojih se slaže objektni repozitorij (eng. object repository).



Slika 2.11: Životni ciklus kod evolucijske domene (preuzeto od Gomaa, 1997, str. 3).

2.4.2 Aktivne biblioteke

Aktivne biblioteke temelje se na konceptu generativnog programiranja (GP). Za razliku od klasičnih biblioteka (Czarnecki et. al., 1998. str. 1), aktivne biblioteke mogu sadržavati metaprograme koji omogućuju generiranje programskog koda specifičnog za pojedinu domenu, optimizacije, debagiranje, profiliranje i testiranje. Takve biblioteke sadrže zajedno implementacijski kod i *metakod* koji može implementirati sintaksna proširenja, izvršiti generiranje programskog koda i primijeniti optimizacije specifične za problemsku domenu.
Tipovi aktivnih biblioteka razlikuju se po vrsti korištenog metaprogramiranja na slijedeće (Czarnecki et. al., 1998. str. 1):

- aktivne biblioteke koje proširuju prevoditelj,
- aktivne biblioteke koje omogućuju podršku za specifične problemske domene i
- aktivne biblioteke koje sadrže *metakod* za proširenje specifične problemske domene na druga područja.

Aktivne biblioteke evoluirale su kasnije, prema (Cilia et al., 2003, str. 169.) u aktivne baze podataka. Današnja istraživanja, provode se prema (Cilia et al., 2003, str. 181.) u smjeru povezivanja aktivnih baza podataka i aspektno orijentiranog programiranja s ciljem stvaranja baza znanja potrebnih za generiranje novih aplikacija.

3 DOSADAŠNJI PRIMJERI UPOTREBE JEZIKA SKRIPATA U GENERATIVNOM PROGRAMIRANJU

Generativno programiranje u početku se uglavnom temeljilo na objektno-orijentiranom programiranju. Nešto kasnije pojavili su se pokušaji da se generativno programiranje poveže i s jezicima skripata, što je najavio Sells (Sells, 2001., str. 1; Sells 2002, str. 1). Tako su se pojavili i neki projekti, među kojima su najznačajniji projekti izrade jezika skripata za generativno programiranje Open Promol, u Litvi, te Codeworker, u Francuskoj. Postoje još neki pokušaji da se jezici skripata približe generativnom programiranju, pa je tako postojeći jezik skripata, Phython, u novijim verzijama uključio tzv. generatore, kao dodatnu mogućnost. Također, pojavila su se i neka hibridna rješenja, gdje se jezici skripata primjenjuju za rješavanje jednog dijela problema unutar generativnog programiranja.

Upotreba jezika skripata trebala bi dovesti do određenih prednosti u odnosu na objektno-orijentirane programske jezike. Te prednosti trebale bi se ostvariti izbjegavanjem određenih slabosti objektno-orijentiranog programiranja, u prvom redu, prema (Ousterhout, 1998. str. 28) krutost objektnog modela, visoki stupanj tipizacije i potreba za fazom prevođenja u razvoju programa ali i korištenjem dobrih svojstava jezika skripata. Od tih svojstava, kao najvažnija mogli bi smo izdvojiti slijedeća:

- mogućnosti jezika skripata u obradi znakovnih nizova (Štuikys, Damaševičius i Ziberkas, 2001., str. 1), (Lemaire, 2003., str. 2)
- mogućnosti povezivanja gotovih komponenti pisanih u ciljnim programskim jezicima (Štuikys, Damaševičius i Ziberkas, 2001., str. 1)
- fleksibilnost sintakse jezika skripata koja proizlazi iz niske razine tipizacije (Štuikys, Damaševičius i Ziberkas, 2001., str. 1), (Ousterhout, 1998. str. 28)

Navedena svojstva iskorištena su dosadašnjim projektima upotrebe jezika skripata u okviru generativnog programiranja, kao što su Open Promol i Codeworker.

3.1 Open Promol

Open Promol (Program Modification Language) je projekt skupine istraživača iz Litve. Kao što mu ime govori, novi programski jezik bavi se modifikacija programskog koda, pisanog u različitim ciljnim programskim jezicima (eng. TL -Target Language). Osnovni ciljevi izrade jezika su, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 1), slijedeći:

- omogućiti specifikaciju modifikacija nad komponentama pisanim u ciljnim programskim jezicima,
- omogućiti specifikaciju povezivanja komponenti pisanih u ciljnim programskim jezicima i
- omogućiti izradu generičkih komponenti i izgradnju sustava za generiranje.

Predviđeno je da novi programski jezik za generiranje bude neovisan o ciljnim programskim jezicima. To znači da taj programski jezik, kao takav, ne proširuje funkcionalnosti tih ciljnih programskih jezika. Umjesto toga, njegova namjena je precizno izraziti pretprogramirane modifikacije programskog koda u ciljnom

programskom jeziku u samo jednoj specifikaciji. Uloga Open Promola je da značajno podigne stupanj apstrakcije i izrazi modifikacije znatno preciznije, šire i fleksibilnije, istovremeno izbjegavajući preveliko poopćenje ciljnog programa (Štuikys, Damaševičius i Ziberkas, 2001., str. 2).

Autori Open Promola su za izradu novog programskog jezika bili su motivirani slijedećim razlozima (Štuikys, Damaševičius i Ziberkas, 2001., str. 1):

- da omoguće ponovnu upotrebljivost programskog koda za dobro definirane problemske domene,
- da osiguraju fleksibilnost potrebnu za reprezentiranje šireg opsega modifikacija i
- da podrže višejezičnu paradigmu u dizajnu sustava.

3.1.1 Glavni koncepti Open Promola

Open Promol temelji se, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 2) na principima modifikacije i principima jezika skripata. Osnovna ideja jezika skripata je, prema (Ousterhout, 1998. str. 28), ta da je programer opskrbljen s kolekcijom upotrebljivih komponenti, nakon čega je potrebno napisati samo malu količinu koda za povezivanje kako bi se ostvarile veze među komponentama. Takav programski kod može biti u raznim oblicima, što ovisi o prirodi i usitnjenosti komponenti, problemskoj domeni i modelu sklapanja. Komponente kao takve su visoko parametrizirani domenski entiteti kao što su vrata, aditori, multiplikatori, itd., poopćeni funkcijama Open Promola (Štuikys, Damaševičius i Ziberkas, 2001., str. 2). Svrha tih funkcija je da izvrše neophodne modifikacije programskog koda u ciljnom programskom jeziku za vrijeme procesa generiranja, sa svrhom dobivanja pojedine instance komponente prilagođene specifičnim potrebama programera (Štuikys, Damaševičius i Ziberkas, 2001., str. 2). Kombinacija modifikacijskih i skriptnih koncepata dovodi do stvaranja parametriziranih sustava (Givargis i Vahid, 2000., str. 98), tj. visoko parametriziranih arhitektura koje se mogu prilagoditi prema zahtjevima snage, performansi i veličine (Štuikys, Damaševičius i Ziberkas, 2001., str. 2).

3.1.2 Osnovna svojstva sintakse i semantike Open Promola

Open Promol je, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 3), funkcionalni jezik i sastoji se od otvorenog skupa eksternih funkcija. Sve modifikacije u specifikaciji su, prema (Štuikys i Damaševičius, str. 2) predstavljene kao kompozicije vanjskih funkcija s programskim kodom u ciljnom programskom jeziku koji treba modificirati. Funkcije su nazvane eksternim jer predstavljaju višu razinu programiranja i imaju parametre, čije vrijednosti su definirane eksterno.

Funkcije jezika Promol imaju slijedeći format, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 3):

@naziv_funkcije [argument_1, argument_2, ... , argument_n]

Znak '@' označava početak funkcije, dok su argumenti uključeni unutar uglatih zagrada. Svaka funkcija, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 3), ima listu argumenata i vraća vrijednost, koja uvijek predstavlja znakovni niz koji sadrži programski kod u ciljnom programskom jeziku. Argumenti mogu biti konstante,

parametri implicitno deklarirani u sučelju specifikacije, izrazi, ili kompozicija drugih funkcija i programskog koda u ciljnom programskom jeziku.

Open Promol orijentiran je na obradu znakovnih nizova i nema atributa za eksplicitnu deklaraciju tipova, tako da su, prema (Štuikys, Damaševičius i Ziberkas, 2001., str. 3) postignuta slijedeća svojstva jezika:

- **Generalizacija**. Moguće je povezivanje sličnih komponenti u jednu generaliziranu ili generičku specifikaciju kroz upotrebu funkcija.
- **Apstrakcija**. Postoje dva stupnja apstrakcije: viša razina odnosi se na sučelje komponente, gdje su definirani parametri specifikacije, dok se niža razina odnosi na implementaciju komponente.
- **Parametrizacija**. Za razliku od sistemskih programskih jezika nije vezana na odgovarajuće tipove (npr. kod predložaka funkcija i klasa u C++), nego koristi znakovne nizove različitih dužina i sadržaja.
- **Odvajanje pogleda** (eng. Separation of concerns). Jasno su odvojene apstrakcije jezika skripata od zadanog ciljnog programskog jezika, sučelje generalizirane specifikacije od njene implementacije i razvoj specifikacije od njene upotrebe.
- **Kompozicija**. Pojedini skup eksternih funkcija korišten u specifikaciji primjenjuje unutarnju kompoziciju koja se temelji na spajanju znakovnih nizova. Korištenje vanjskih modula (pomoću @macro) omogućuje napredniju kompoziciju.
- **Generiranje**. Postoje specijalizirane funkcije za generiranje znakovnih nizova zadane strukture.
- **Proširivost**. Open Promol je otvoreni skup neovisnih vanjskih funkcija, te omogućuje uvođenje novih.
- **Modifikacija**. Open Promol omogućuje modifikacije ciljnog programa korištenjem uobičajenih apstrakcija strukturnog programiranja (npr. if, for, case itd.).
- **Neovisnost o ciljnom programskom jeziku**. Open Promol tretira program u ciljnom programskom jeziku kao običan tekst bez posebnog značenja i koristi jezične mehanizme koji su neovisni o sintaksi bilo kojeg ciljnog programskog jezika.
- **Brzo prototipiranje**. Jezični procesor implementira to svojstvo kroz generiranje pojedine ili svih instanci definiranih specifikacijom.
- **Dokumentiranje**. Korisnički orijentirana pitanja u sučelju i komentari u tijelu specifikacije omogućuju izražavanje u prirodnom jeziku.

3.1.3 Primjeri

Slijedeći primjer demonstrira upotrebu jezika Open Promol za generiranje zadanih znakovnih nizova, prema (Štuikys i Damaševičius, 2000, str. 4):

Generirani znakovni nizovi:

X1, X2, X3, X4, *(1)*

Y (1) + Y(2) + Y(3) *(2)*

A10 AND A11 AND A12 AND A13 AND A14 *(3)*

Izrazi u Open Promolu:

```
@gen [4,{,},{x},1],          (1)
@gen [3,{)+},{Y(},1]),       (2)
@gen [5,{AND},{A},10].       (3)
```

Nešto složeniji primjer je slijedeći i pokazuje kako se integriraju programski kod predloška i pozivi funkcija za generiranje:

```
$
"Enter a number of inputs :" {2,3,4,5,6,7,8} num:=2;
"Enter a function's name :" {AND,OR,XOR,NOR,NAND,XNOR} f:=OR;
"Enter the initial value :" {0,1,2,3,4,5,6,7,8,9,10} init:=10;
"Do you want to use a generic delay (0-no, 1-yes):" {0,1} use:=1;
use=1 "Enter a delay constant in ns. : " {1,2,3,4,5} delay:=1;
$
ENTITY  GATE_@sub[ f]  IS
  @if[ use,{GENERIC ( T : TIME := @sub[ delay] NS);}]
   PORT ( @gen[ num,[ , ],{X},init]  : IN  BIT;
              Y : OUT BIT );
END  GATE_@sub[ f];

ARCHITECTURE  BEHAVE_GATE_@sub[ f]  OF  GATE_@sub[ f]  IS
BEGIN
  Y <= @gen[ num, ( @sub[ f] ),{X},init] @case[ use+1,{ },{AFTER T}];
END  BEHAVE_GATE_@sub[ f];
```

Slika 3.1: Primjer integracije predloška koda i funkcija za generiranje (Štuikys i Damaševičius, 2000, str. 10)

Na slici 3.1. vidimo upotrebu znaka '@' koji označava pozive funkcija. Povratne vrijednosti funkcija (znakovni nizovi) zamjenjuju pozive u generiranom programskom kodu.

3.2 Codeworker

Codeworker predstavlja projekt u okviru Open Source zajednice za izradu alata za obradu znakovnih nizova (eng. parsing tool) i generatora izvornog koda. Voditelj projekta je Cedric Lemaire (Lemaire, 2003., str. 2). Razvijen novi jezik skripata, čija sintaksa je u osnovi preuzeta od jezika srodnih C-u, ali je prilagođena za potrebe generiranja koda.

Izrađeni alati, temeljeni na novom jeziku skripata, omogućuju generiranje programskog koda u raznim programskim jezicima, i to na slijedeće načine (Lemaire, 2003., str. 2):

- **generiranje** korištenjem skripata, sličnih onima u jezicima JSP ili PHP, za izradu izlaznih datoteka,
- **ekspanzija** (eng. expansion) se koristi kada treba generirati male dijelove postojećih datoteka. Točke gdje se umeće programski kod nazivaju se markeri (eng. markers).
- **mod za prevođenje** (eng. translation mode) koristi se kad su potrebni i obrada znakovnih nizova i generiranje izvornog koda da bi se dobila izlazna datoteka. Moguće su dvije osnovne vrste prevođenja:
    - **prevođenje izvornog koda u izvorni kod** (eng. source to source translation). U tom slučaju datoteka se prepisuje u drugu datoteku s novom sintaksom. Primjer prevođenje dokumenta u LaTeX formatu u HTML i
    - **transformacija programa** (eng. program transformation). Izvorni kod se mijenja radi optimizacije ili promjene svojstava pojedinih dijelova. Na primjer, skripta može dodati pojedine dijelove programskog koda na označenim pozicijama.

3.2.1 Svojstva jezika

Codeworker je jezik skripata koji se implementira kao interpreter. Obrada znakovnih nizova, predstavljanje podataka i generiranje izvornog koda integrirani su unutar istog jezika, a sintaksa se može prilagoditi za pojedine od tih zadataka (Lemaire, 2003., str. 7). Ostala specifična svojstva jezika su slijedeća (Lemaire, 2003., str. 7):

- koristi se oznaka '@' koja omogućuje prijelaz između generiranja teksta i interpretacije skripti. Osim te oznake mogu se također koristiti i oznake '<%' i '%>'.
- sve potrebne funkcije za čitanje tokena, te izbjegavanje praznina i komentara
- pisanje skripti za obradu znakovnih nizova po uzoru na BNF (pogodnije od čitanja tokena) što omogućuje promjenu sintakse
- strukturirani podaci kao stabla koja se popunjavaju na vrlo prirodan način
- sintaksa po uzoru na klasične programske proceduralne jezike, kao što su C++ ili Pascal (deklaracije funkcija, iteracija listi, upravljanje pomoću try/catch i sl.),
- integrirani program za pronalaženje pogrešaka (eng. debugger),
- alat za profiliranje koji mjeri pokrivenost skripti i utrošeno vrijeme.

Zahvaljujući tim svojstvima ostvarene su slijedeće dobrobiti za programere (Lemaire, 2003., str. 7):

- predlošci skripata (namijenjeni generiranju izvornog koda) odražavaju sposobnosti programera na način da ih se učini ponovo iskoristivim,
- što više koncepata programer uključi u modeliranje, to više rješenja arhitekture ili dizajna aplikacija postaje moguće,
- dobar formalni jezik za modeliranje omogućuje da nikada nije potrebno dva puta pisati isti kod, nego umjesto toga se većina koda implementira automatski.

3.2.2 Primjer

U slijedećem primjeru koristi se skripta predložak za generiranje programskog koda u jeziku Java (Lemaire, 2003., str. 14):

```
// file "Scripts/Tutorial/GettingStarted/Tiny-JAVA.cwt":
    1 package tiny;
    2
    3 public class @name@ @
    4 if existVariable(parent) {
    5     @ extends @parent.name@ @
    6 }
    7 @{
    8     // attributes:
    9 @
```

Primjer generiranog programskog koda mogao bi izgledati ovako (Lemaire, 2003., str. 15):

```
// file "Scripts/Tutorial/GettingStarted/tiny/D.java":
    package tiny;
```

```
public class D {
   // attributes:
   private A _a = null;
   private java.util.ArrayList/*<C>*/ _c = null;

   //constructor:
   public () {
   }
```

U primjeru možemo vidjeti da su oznake unutar znakova '@' zamijenjene odgovarajućim programskim kodom. Također, moguće je i uvjetovano generiranje, što je ovdje zadano izrazom *if existVariable(parent)* - (funkcija 'existVariable' ovdje provjerava postojanje tijela klase).

3.3 Svojstva dosadašnjih primjera upotrebe jezika skripata u okviru generativnog programiranja

Na temelju proučenih projekata možemo izdvojiti slijedeća zajednička svojstva:

- u projektima Open Promol i Codeworker razvijeni su novi skriptni jezici, prilagođeni za generativno programiranje,
- oba jezika koriste predloške programskog koda (metaprograme) koji sadrže funkcije za generiranje (u oba jezika označavaju se znakom '@'), uz korištenje principa modifikacije,
- u oba jezika moguće je generirati programe u različitim programskim jezicima, ali i raznih drugih znakovnih nizova, koji ne moraju predstavljati programe.

Preostali su i određeni problemi, koji će se obrađivati u okviru ovog rada:

- za sada još nisu ponuđeni grafički modeli za modeliranje generatora,
- pozivi funkcija generiranja integrirani s kodom u ciljnom programskom jeziku, što onemogućuje korištenje istog generatora za generiranje programskog koda u različitim programskim jezicima, odnosno, općenito generiranje različitih inačica koda,
- složena sintaksa poziva funkcija generiranja,
- složena specifikacija aplikacija,
- implementacija aspekata je nejasna.

Osim projekata Open Promol i Codeworker, mogli bi smo identificirati još neke primjere u kojima se jezici skripata koriste u okviru generativnog programiranja, ali je njihova uloga pomoćna ili se odnosi samo na pojedine dijelove generativnih sustava, kao npr. DataScript. DataScript (Back, 2002., str. 66-77) predstavlja, prema jezik skripata za opis i manipulaciju binarnih podataka kao tipova. Primjer upotrebe je mapiranje DataScriptovih tipova na klase u jeziku Java. DataScript-ov prevoditelj generira Java klasu za svaki složeni tip, što uključuje pristupnu metodu za svako polje.

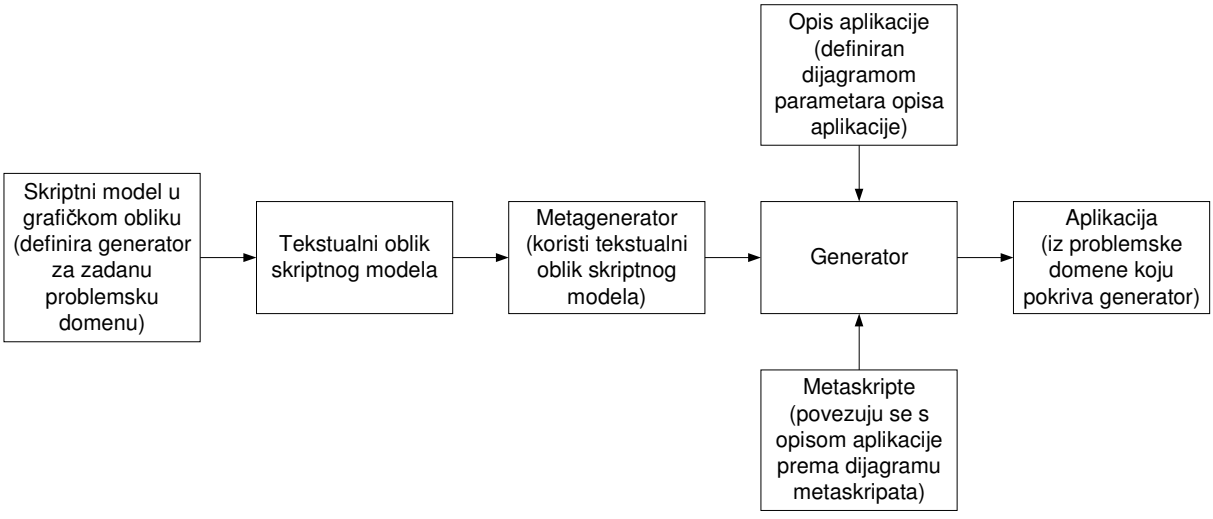
4 SKRIPTNI MODEL GENERATORA APLIKACIJA

Skriptni model razvijen je za potrebe generativnog programiranja temeljenog na jezicima skripata. Za razliku od objektnog modela, orijentiran je na definiranje zadanih, specifičnih aspekata budućih aplikacija unutar zadane problemske domene, a ne na sve funkcionalnosti aplikacija, jer se te definiraju na nižoj razini (u predlošcima koda aplikacija, koje u okviru skriptnog modela nazivamo metaskriptama). Aspekti prema (Kinczales et al., 1997., str. 1) predstavljaju svojstva koja "presijecaju osnovnu funkcionalnost sustava" (Kinczales et al., 1997., str. 1), odnosno svojstva koja nisu vezana za pojedine organizacijske jedinice programa kao što su funkcije ili klase, nego se mogu pojavljivati u različitim dijelovima aplikacija, prema (Lee K.W.K, 2002., str. 1). Zbog toga je bilo potrebno razviti model povezivanja, koji u osnovi predstavlja model spojnih točaka (eng. join points model; u okviru skriptnog modela to je dijagram metaskripata).

Skriptni model sastoji se od dva grafička dijagrama (ili ekvivalentnih tekstualnih specifikacija) i kao takav jednostavniji je od modela temeljenih na UML-u. Međutim, kao što će biti pokazano u poglavlju 4.5, postoji kompatibilnost između tih dvaju modela. Jezici skripata ne temelje se na klasama i objektima, no, njihova osnovna svojstva, enkapsulacija, nasljeđivanje, skrivanje podataka i polimorfizam mogli bi se postići na višoj razini skriptiranja (eng. scripting - izrada programa u jezicima skripata). Skripte više razine nazivat ćemo *metaskriptama*. Metaskripte predstavljaju predloške (eng. template) za generiranje programskog koda u različitim programskim jezicima.

Postoje, naravno, i bitne razlike između objektnog i skriptnog modela, uključujući i tu da je skriptni model temeljen na aspektima, dok modeliranje aspekata još uvijek predstavlja problem u okviru objektnog modela (vidi: poglavlje 2.2.2.3. Modeliranje u okviru aspektno-orijentiranog programiranja). Također, objektni model definira pojedinu aplikaciju, dok skriptni model definira generator aplikacija za zadanu problemsku domenu.

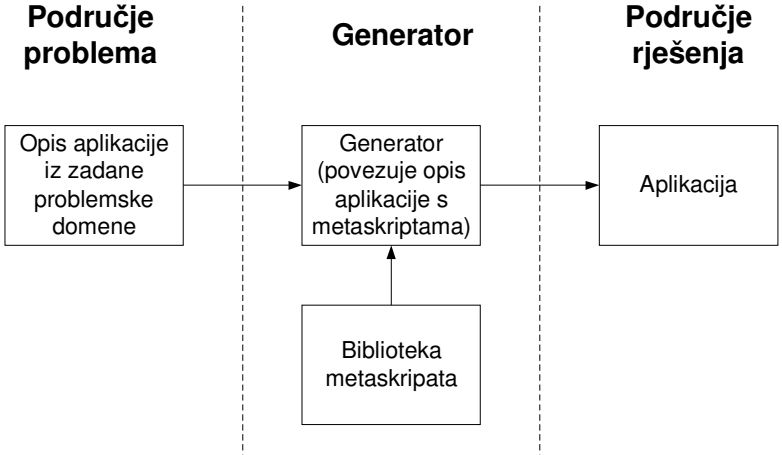
Proces razvoja generatora aplikacija možemo prikazati slijedećim dijagramom (slika 4.1):



Slika 4.1: Proces razvoja generatora aplikacija prema konceptu temeljenom na jezicima skripata

Generator aplikacija koristi tekstualnu specifikaciju čija struktura je zadana dijagramom parametara opisa aplikacije. Tekstualnu specifikaciju ćemo u daljnjem tekstu nazivati opis aplikacije

Programski jezik ciljne (generirane) aplikacije neovisan je o skriptnom modelu, odnosno, definiran je metaskriptama. Pojedina aplikacija generira se povezivanjem opisa aplikacije s metaskriptama (slika 4.2). Način povezivanja definiran je dijagramom metaskripata.



Slika 4.2: Generiranje aplikacije iz zadane problemske domene

Opis aplikacije sadrži sva specifična svojstva aplikacije po kojima se ta aplikacija razlikuje od ostalih unutar problemske domene koju pokriva generator. Sva zajednička svojstva različitih aplikacija unutar problemske domene generatora definirana su unutar metaskripata. Time je postignut bolji pregled nad cjelinom aplikacije. Također, pojednostavljeno je održavanje aplikacija koje se sastoje od više programskih modula, ponekad pisanih i u različitim programskim jezicima, budući da se promjene u opisu aplikacije za vrijeme generiranja ažuriraju u svim modulima. Taj problem ažuriranja više programskih modula osobito je izražen kod web aplikacija pisanih u jezicima skripata, jer se one tipično sastoje od većeg broja programskih modula i drugih datoteka (npr. forme za unos/ispravak podataka i drugi dokumenti u HTML-u).

4.1 Razine generiranja

Generatori aplikacija namijenjeni su generiranju aplikacija iz zadane problemske domene. Uočeni su glavni problemi pri izradi generatora aplikacija:

- **kako poopćiti generator aplikacija**, tako da može generirati aplikacije iz što šire problemske domene (to se prema skriptnom modelu postiže uvođenjem višerazinskih generatora) i
- **kako automatizirati postupak izrade generatora**. U osnovi su moguća dva pristupa koji će biti detaljnije razrađeni u okviru ovog rada:

- izradom općeg generatora, koji bi se pomoću odgovarajućih specifikacija (skriptnog modela) prilagodio za pojedine problemske domene (generički pristup) ili
- pomoću metageneratora, odnosno generatora koji na temelju odgovarajuće specifikacije (skriptnog modela) daje na izlazu generator aplikacija iz zadane problemske domene (generativni pristup).

Bez obzira na pristup (generički ili generativan), za izradu generatora prema predloženom skriptnom modelu potrebne su slijedeće informacije:

- opis sintakse jezika za zadavanje opisa aplikacija (problemske domene) i
- odgovarajući predlošci programskog koda generatora aplikacija

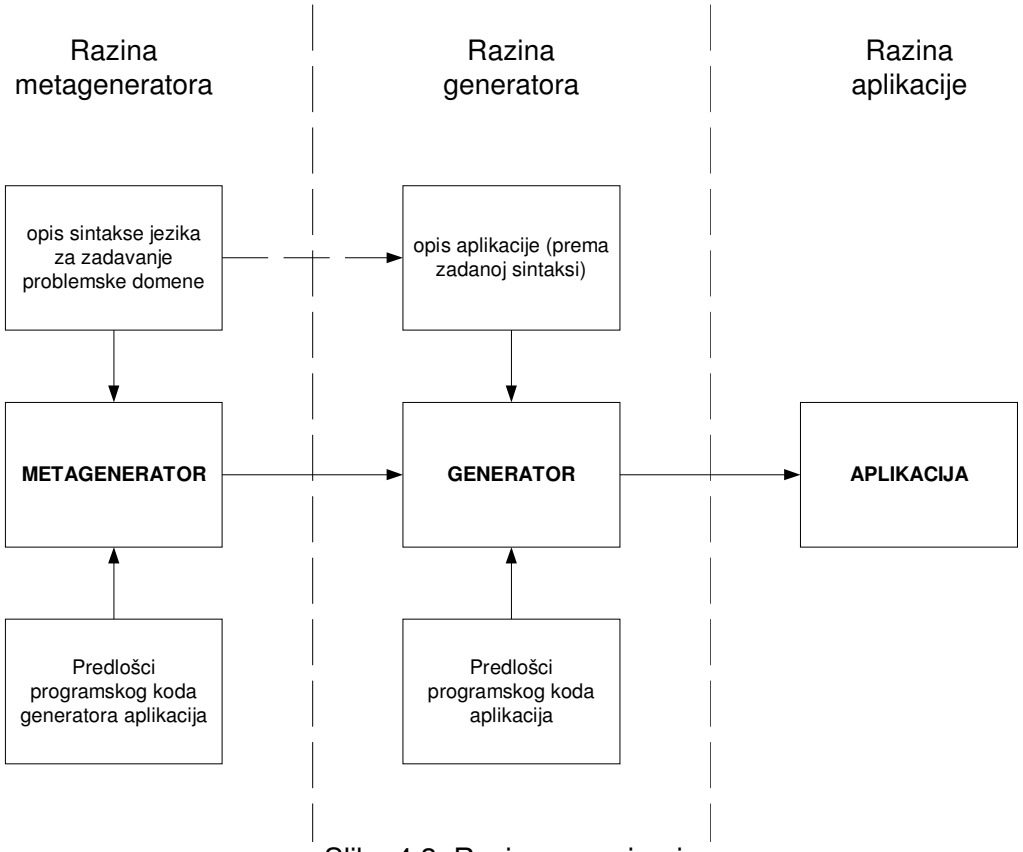
Pojedini generator za generiranje aplikacija treba slijedeće informacije (slika 4.2):

- opis aplikacije, zadan prema definiranoj sintaksi (čiji opis koristi metagenerator) i
- predlošci programskog koda aplikacija.

Razlikujemo, dakle, tri razine generativnog programiranja prema predloženom konceptu, temeljenom na jezicima skripata, koji je predmet ovog rada:

1. razina metageneratora (poopćena razina generatora aplikacija)
2. razina generatora aplikacija (poopćena razina aplikacije)
3. razina aplikacije (ciljna razina)

Odnos između te tri razina generiranja možemo vidjeti na slijedećem dijagramu (slika 4.3):



Slika 4.3: Razine generiranja

4.2 Dijagrami skriptnog modela

Skriptni model generatora aplikacija je grafički model koji definira zadana svojstva (aspekte) prema kojima će se pojedina aplikacija razlikovati od ostalih aplikacija iz iste problemske domene, te način distribucije tih svojstava unutar aplikacije. To se postiže pomoću dva dijagrama:

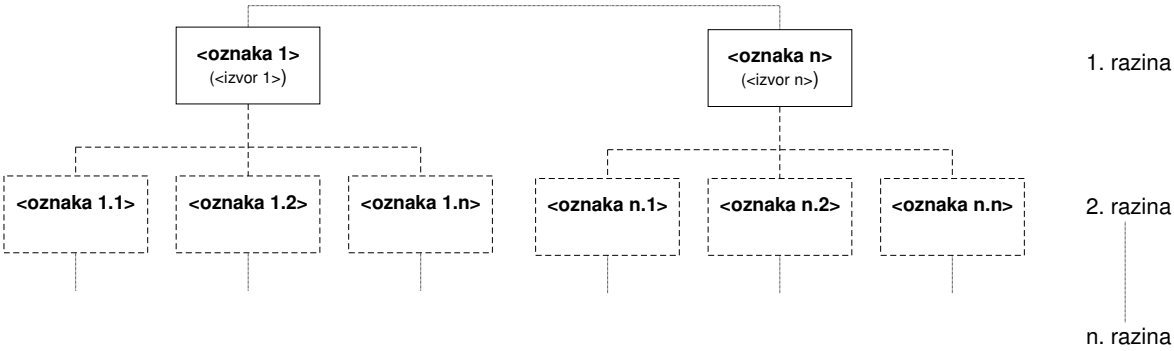
- **dijagram parametara opisa aplikacije** - definira zadana svojstva (aspekte) buduće aplikacije unutar zadane problemske domene i definira strukturu opisa aplikacije, na temelju kojeg će se pomoću odgovarajućeg generatora generirati aplikacija.
- **dijagram metaskripata** - definira distribuciju pojedinih specifičnih svojstava, zadanih dijagramom parametara opisa aplikacije, unutar aplikacije, te njihovo povezivanje s predlošcima programskog koda (metaskriptama).

4.2.1 Dijagram parametara opisa aplikacije

Dijagram parametara opisa aplikacije je hijerarhijski dijagram koji definira zadana svojstva aplikacije. Svojstva su definirana parametrima koji se koriste za opis aplikacije iz domene generatora. Parametri opisa aplikacije imaju hijerarhijsku strukturu i definiraju pojedinu aplikaciju unutar zadane problemske domene, obuhvaćene generatorom, a mogu se odnositi na:

- definiciju podataka i
- definiciju vrste obrade.

Dijagram parametara opisa aplikacije sadrži oznake (engl. tag) i pridružene izvore podataka, kao što se vidi na slici 4.4.



Slika 4.4: Dijagram parametara opisa aplikacije

Oznakama se u opisu aplikacije pridružuju vrijednosti, na temelju kojih se aplikacija generira, dok izvori predstavljaju strukture podataka u koje će se upisivati konkretne vrijednosti iz opisa aplikacije. Moguće su dvije vrste podataka (označavaju se znakovima '$' i '@', po uzoru na jezik Perl):

- **pojedine varijable**. Pojedine varijable označavaju se znakom '$'. Oznake s izvorom definiranim pojedinom varijablom ne mogu se dalje granati, odnosno, sadrže samo prvu razinu.

- **polja s neograničenim brojem elemenata**. Označavaju se znakom '@'.

Ovdje treba primijetiti da se izvori navode samo na prvoj razini dijagrama parametara opisa aplikacije, jer su zajednički za pojedinu granu dijagrama.

Također, pri imenovanju oznaka u dijagramu parametara opisa aplikacije mogu se izvršiti racionalizacije, odnosno moguće je definirati oznaku čiji naziv se sastoji od više od jednog dijela, uz korištenje znakova '_' i '.' odnosno:

<naziv zajedničkog dijela>_<naziv specifičnog dijela>

odnosno

<naziv zajedničkog dijela>.<naziv specifičnog dijela>

Time se smanjuje broj potrebnih razina u dijagramu metaskripata. Tipičan primjer je rad s podacima različitih tipova - definiraju se različito, ali se neke vrste obrada mogu na jednak način provoditi za različite tipove podataka.

Odgovarajuća struktura opisa aplikacije proizlazi iz dijagrama parametara opisa:

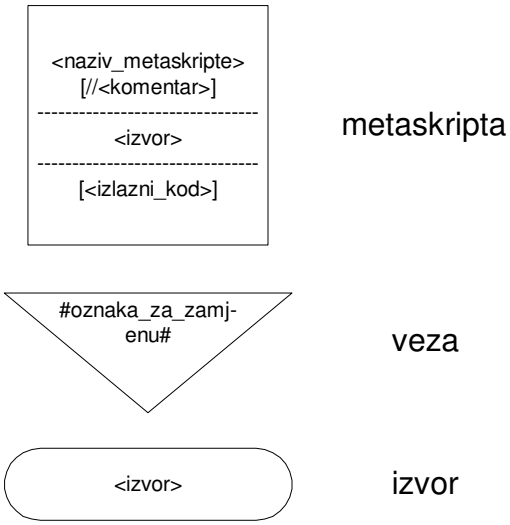
<oznaka 1>: <vrijednost 1>
<oznaka 1.1>: <vrijednost 1.1>
<oznaka 1.2>: <vrijednost 1.2>
<oznaka 1.n>: <vrijednost 1.n>
<oznaka n>: <vrijednost n>
.
.
.
<oznaka n.1>: <vrijednost n.1>
<oznaka n.2>: <vrijednost n.2>
<oznaka n.n>: <vrijednost n.n>

Sve oznake se unutar aplikacije zadaju opcionalno (mogu se izostaviti, odnosno pojaviti se proizvoljan broj puta). Također, moguće su oznake kojima se ne pridružuju vrijednosti.

4.2.2 Dijagram metaskripata

Dijagram metaskripata definira distribuciju pojedinih specifičnih svojstava, zadanih dijagramom parametara opisa aplikacije, unutar aplikacije, te njihovo povezivanje s predlošcima programskog koda (metaskriptama). Drugim riječima, dijagram metaskripata predstavlja shemu spajanja aplikacije.

Koriste se tri osnovna elementa od kojih se sastoje dijagrami (slika 4.5):



Slika 4.5: Elementi dijagrama metaskripata

Metaskripta predstavlja osnovni predložak na temelju kojeg se generira njegova implementacija - programski kod u ciljnom programskom jeziku. Unutar pravokutnika upisuju se slijedeći podaci:

- naziv metaskripte
- komentar označen s dvije kose crte (opcionalno)
- izvor (naziv datoteke ili drugog medija koji sadrži kod metaskripte)
- izlazni kod (naziv datoteke ili drugog medija koji sadrži generirani programski kod; opcionalno)

Metaskripte sadrže oznake za zamjenu - oznake koje se zamjenjuju odgovarajućim podacima ili programskim kodom. Oznake se u dijagramu metaskripata zadaju pomoću elementa veze.

Veza povezuje metaskriptu s izvorom podataka za zamjenu, te (opcionalno) s jednom ili više metaskripata niže razine, koje se koriste za generiranje zamjenskog koda. Predstavlja se trokutom s jednim od vrhova prema dolje, unutar kojeg se navodi naziv oznake za zamjenu. Sama zamjena koda može se izvršiti na jedan od slijedeća dva načina:

- direktna zamjena oznake izvornim podatkom (ako nema metaskripte niže razine) i
- zamjena oznake metaskriptom niže razine (koja također može sadržavati vlastite veze i izvore).

Izvor predstavlja pojedini parametar zadan odgovarajućom oznakom u okviru dijagrama parametara opisa aplikacije. Bitno je da se kao izvori mogu zadati skupni parametri (imaju svoju strukturu, odnosno, granaju se) - u kojem slučaju je potrebno njihovu daljnju primjenu definirati na nižim razinama dijagrama metaskripata, odnosno pojedinačni parametri (ne granaju se).
Izvor se zadaje kao pojedini parametar (definiran odgovarajućom oznakom) iz dijagrama parametara opisa aplikacije:

<oznaka> - naziv oznake

Naziv oznake može se sastojati samo od zajedničkog dijela. U tom slučaju naziv oznake završava sa znakom '_' ili '.' :

<oznaka>_ ili <oznaka.>

Primjer:

<polje_> - obuhvaća sve oznake čiji naziv počinje s 'polje_' (npr. polje_broj, polje_real ili polje_znak)

Također, neki izvori zahtijevaju predobradu pomoću odgovarajuće funkcije, što se označava na slijedeći način:

&<naziv_funkcije(<parametri>)>

pri čemu parametri predstavljaju polja s neograničenim brojem elemenata, koja se navode na prvoj razini u dijagramu parametara opisa aplikacije.

Kod zadavanja izvora moguća je racionalizacija u slučajevima kada se za više različitih parametara definiranih u dijagramu parametara opisa aplikacije koristi ista metaskripta. U tom slučaju dovoljno je kod izvora upisati samo zajednički dio naziva izvora, koji završava znakom '_' ili '.' :

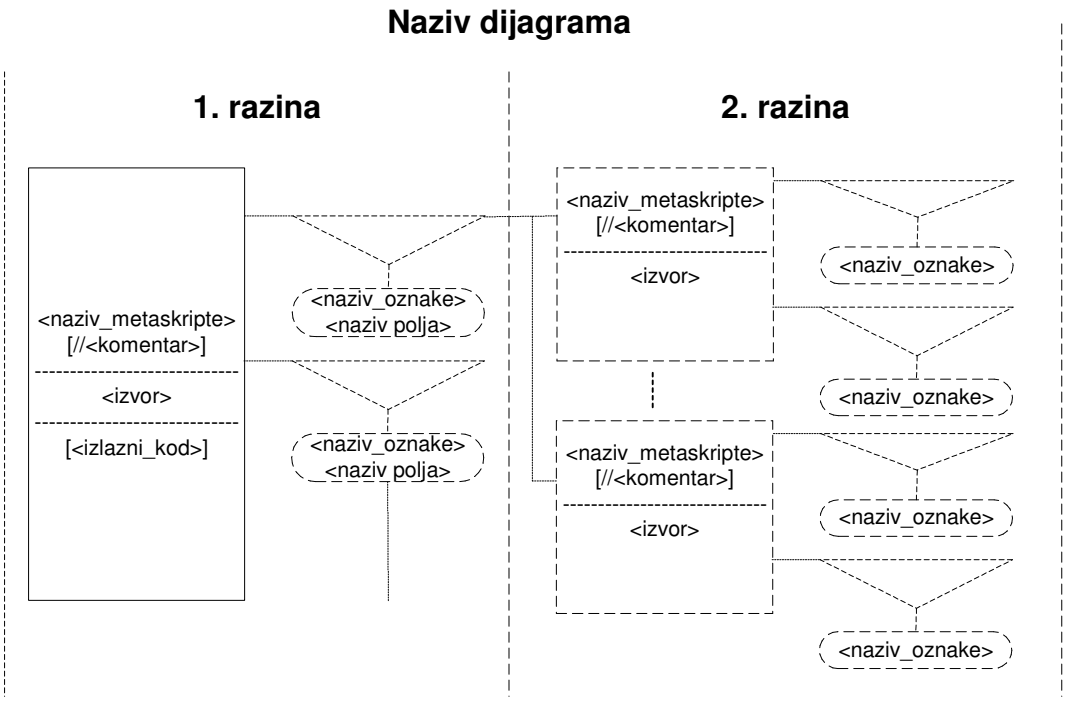
<zajednički dio naziva izvora>_

ili

<zajednički dio naziva izvora>.

4.2.2.1 Raspored elemenata dijagrama metaskripata

Elementi dijagrama metaskripata raspoređuju se u vertikalne stupce, koje možemo usporediti s plivačkim stazama. Svaki od stupaca predstavlja odgovarajuću razinu metaskripata, kao što se može vidjeti na slici 4.6:

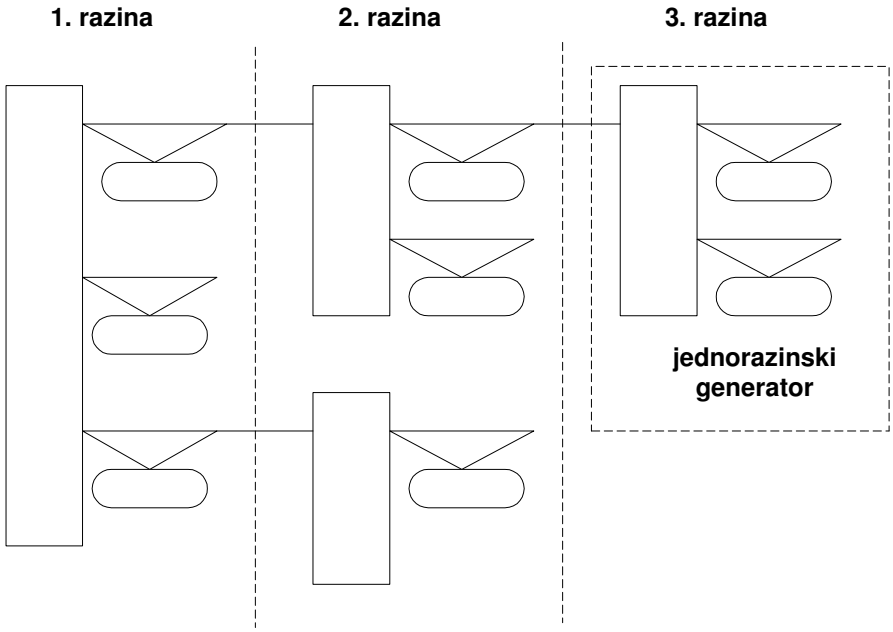


Slika 4.6: Raspored elemenata na dijagramu metaskripata

Na slici 4.6 vidi se da element 'veza' povezuje metaskripte s pridruženim izvorima podataka, odnosno predlošcima niže razine. Veza s predlošcima niže razine je tipa 1: 0..N .

Broj razina dijagrama metaskripata određuje razinu generatora, tako da razlikujemo:

- jednorazinski generator - jednostavan generator koji se sastoji od samo jednog predloška i čiji dijagram metaskripata sadrži samo jednu razinu i
- višerazinski generator - složeni generator koji uključuje više predložaka koda na različitim razinama, odnosno, čiji dijagram metaskripata sadrži više razina. Svaka grana u dijagramu metaskripata definira odgovarajući generator niže razine. Najniža razina svake grane dijagrama metaskripata definira jedan jednostavan jednorazinski generator (slika 4.7). Višerazinski generator dobije se superpozicijom više jednorazinskih generatora.



Slika 4.7: Na najnižoj razini svake grane dijagrama metaskripata definiran je jedan jednostavan jednorazinski generator

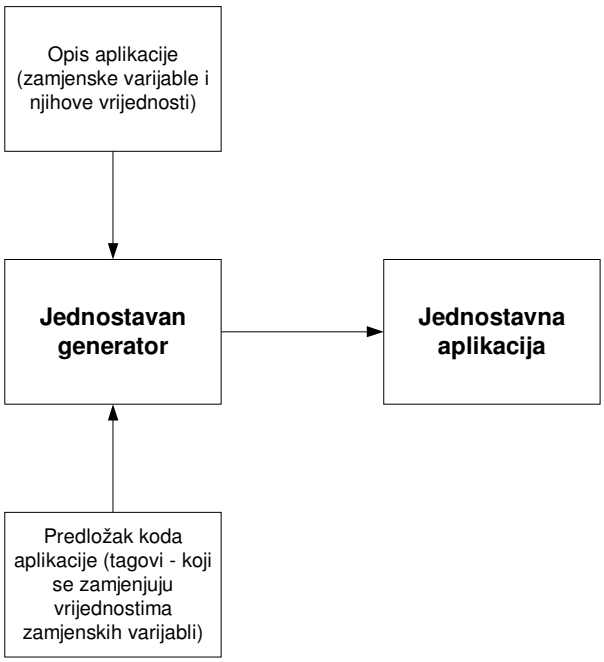
4.3 Generatori prema skriptnom modelu

U ovom dijelu obrađen je razvoj jednog jednostavnog jednorazinskog generatora i jednog višerazinskog generatora prema predloženom skriptnom modelu. Izdvojene su funkcije generiranja, te njihova implementaciju u jeziku Perl. Također, uočene su korisne osobine jezika skripata, kao što je Perl, za realizaciju generatora aplikacija.

4.3.1 Jednostavan generator

Jednostavan generator koristi samo jedan predložak programskog koda (metaprogram) u kojem su varijabilni dijelovi zamijenjeni odgovarajućim oznakama. Svaka pojedina oznaka ima svoju zamjensku varijablu u opisu aplikacije, gdje je navedena i njena vrijednost. Takav jednostavan generator omogućuje odnos između zamjenske varijable i odgovarajuće oznake 1:1..N, odnosno, pojedina oznaka može se unutar pojedinog predloška programskog koda upotrijebiti više puta.

Odgovarajući generator, temeljen na jezicima skripata (u primjeru se koristi Perl) možemo prikazati slijedećim dijagramom (slika 4.8):



Slika 4.8: Jednostavan generator

Primjer: Jednostavan generator programskog koda u jeziku C++

Jednostavan generator ima zadatak učitati od korisnika vrijednosti triju varijabli, različitih tipova, te ispisati njihove vrijednosti.

Koraci rješavanja problema:

1. Utvrditi potrebne zamjenske varijable i njihove oznake (tablica 4.1):

zamjenska varijabla	**oznaka**
prva	#prva#
tip_prve	#tip_prve#
druga	#druga#
tip_druge	#tip_druge#
treca	#treca#
tip_trece	#tip_trece#

Tablica 4.1: Zamjenske varijable i njihove oznake za jednostavan generator

2. Izraditi predložak aplikacije u ciljnom programskom jeziku (ovdje C++):

```
#include <iostream.h>

#tip_prve# #prva#;
#tip_druge# #druga#;                    deklaracije varijabli
#tip_trece# #treca#;

void main(){
        //unos vrijednosti varijabli
        cout << "#prva# = ";
        cin >> #prva#;
        cout << "#druga# = ";           unos vrijednosti varijabli
        cin >> #druga#;
        cout << "#treca# = ";
```

```
        cin >> # treca#;
        // ispis vrijednosti varijabli
        cout << "--------------------" << endl;
        cout << "#prva# (#tip_prve#) = ";
        cout << #prva# << endl;
        cout << "#druga# (#tip_druge#) = ";      ispis vrijednosti varijabli
        cout << #druga# << endl;
        cout << "#treca# (#tip_trece#) = ";
        cout << #treca# << endl;
}
```

3. Definirati vrijednosti zamjenskih varijabli (opis aplikacije):

```
prva:cjelobrojna
tip_prve:int
druga:realna
tip_druge:float
treca:znakovna
tip_trece:char
```

4. Izraditi odgovarajući generator u jeziku skripata (ovdje Perl):

```
open (dat,"<jednostavan generator.predlozak");
@predlozak=<dat>;              Učitavanje predloška jednostavne aplikacije
close (dat);
```

```
open (dat,"<opis jednostavne aplikacije.txt");
@opis=<dat>;                   Učitavanje opisa jednostavne aplikacije
close (dat);
```

```
@o=@opis;
foreach $line(@o){ #petlja za čitanje opisa aplikacije

$line=~ s/\n//; #uklanjanje znaka za kraj retka
#čitanje naziva prve varijable
if ($line =~ /prva:/){
$prva=$line;
$prva=~ s/prva://;}
#čitanje tipa prve varijable       Čitanje vrijednosti zamjenskih
if ($line =~ /tip_prve:/){         varijabli iz opisa aplikacije
$tip_prve=$line;
$tip_prve=~ s/tip_prve://;}
#čitanje naziva druge varijable
if ($line =~ /druga:/){
$druga=$line;
$druga=~ s/druga://;}
#čitanje tipa druge varijable
if ($line =~ /tip_druge:/){
$tip_druge=$line;
$tip_druge=~ s/tip_druge://;}
#čitanje naziva treće varijable
if ($line =~ /treca:/){
$treca=$line;
$treca=~ s/treca://;}
#čitanje tipa treće varijable
if ($line =~ /tip_trece:/){
$tip_trece=$line;
```

```
$tip_trece=~ s/tip_trece://;}

} # kraj petlje za čitanje opisa aplikacije
```

```
open (dat,">aplikacija.cpp");
@p=@predlozak;
foreach $line(@p){ #petlja za čitanje predloška aplikacije
#zamjene oznaka vrijednostima zamjenskih varijabli
$line=~ s/#prva#/$prva/;
$line=~ s/#tip_prve#/$tip_prve/;
$line=~ s/#druga#/$druga/;
$line=~ s/#tip_druge#/$tip_druge/;
$line=~ s/#treca#/$treca/;
$line=~ s/#tip_trece#/$tip_trece/;
#upis programskog retka u ciljnu aplikaciju
print dat $line;
} # kraj petlje za čitanje predloška aplikacije
close (dat);
```

Pridruživanje vrijednosti zamjenskih varijabli oznakama u predlošku aplikacije i kreiranje ciljne aplikacije

Generirani programski kod u C++ izgleda ovako:

```
#include <iostream.h>

int cjelobrojna;
float realna;
char znakovna;
```

deklaracija varijabli

```
void main(){
        //unos vrijednosti varijabli
        cout << "cjelobrojna = ";
        cin >> cjelobrojna;
        cout << "realna = ";
        cin >> realna;
        cout << "znakovna = ";
        cin >> znakovna;
        // ispis vrijednosti varijabli
        cout << "--------------------" << endl;
        cout << "cjelobrojna (int) = ";
        cout << cjelobrojna << endl;
        cout << "realna (float) = ";
        cout << realna << endl;
        cout << "znakovna (char) = ";
        cout << znakovna << endl;
}
```

unos vrijednosti varijabli

ispis vrijednosti varijabli

Kod jednostavnog generatora možemo primijetiti da se pojedine oznake koriste više puta za odgovarajuće zamjenske varijable, međutim, ostaje osnovno ograničenje jednostavnog generatora, a to je slijedeće:

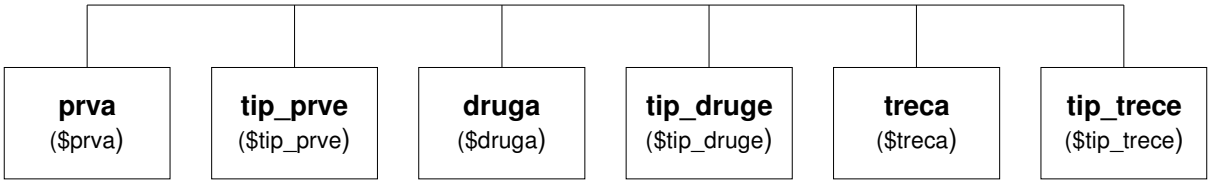
- odnos 1:1 između zamjenskih varijabli i oznaka u predlošcima onemogućuje implementaciju složenijih struktura u opisu aplikacije, te više razina predložaka programskog koda.

4.3.1.1 Skriptni model jednostavnog generatora

Jednostavni generator aplikacija možemo modelirati pomoću odgovarajućih dijagrama:

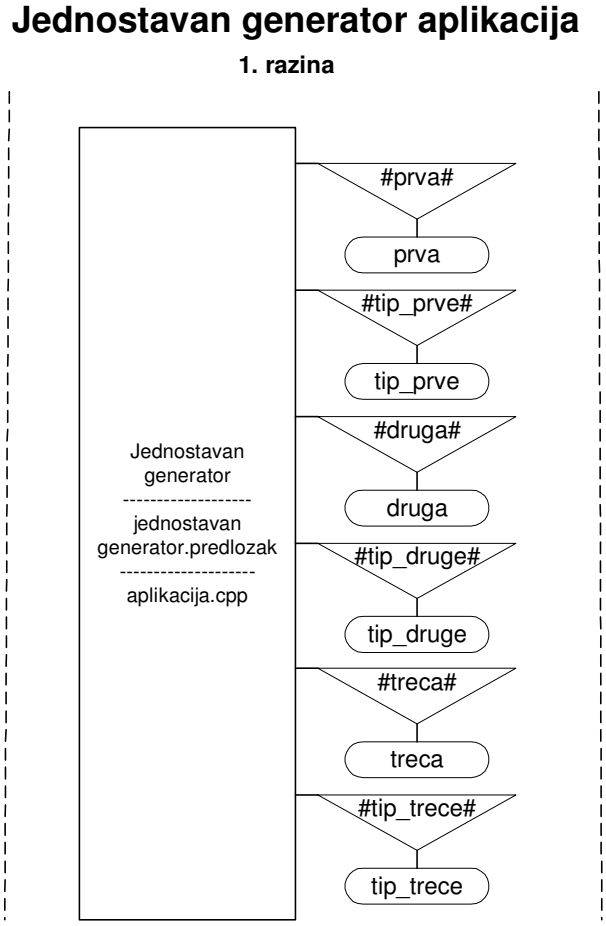
- dijagrama parametara opisa aplikacije i
- dijagrama metaskripata

Dijagram parametara opisa aplikacija za jednostavni generator aplikacija sadrži samo jednu razinu (slika 4.9):



Slika 4.9: Dijagram parametara opisa jednostavnog generatora aplikacija iz primjera

Dijagram metaskripata za aplikaciju iz primjera također bi sadržavao samo jednu razinu (slika 4.10):



Slika 4.10: Dijagram metaskripata jednostavnog generatora aplikacija iz primjera

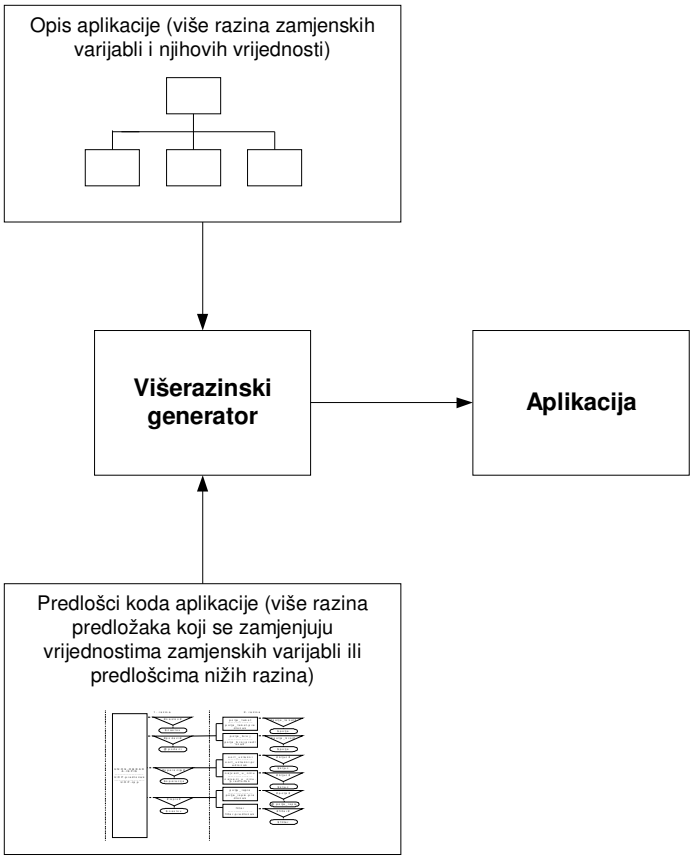
Oznake iz dijagrama parametara opisa aplikacije raspoređene su u dijagramu metaskripata kao izvori, koji se preko elementa veze povezuju s metaskriptom.

4.3.2 Višerazinski generator

Višerazinski generator aplikacija treba omogućiti generiranje složenijih aplikacija, odnosno:

- generiranje aplikacija koje se moraju definirati pomoću više razina opisa (definira se pomoću dijagrama parametara opisa aplikacije) i više razina predložaka koda (definira se pomoću dijagrama metaskripata).

Takav generator mogao bi se prikazati slijedećim dijagramom (slika 4.11):



Slika 4.11: Višerazinski generator

Kod višerazinskog generatora aplikacija problem povezivanja zamjenskih varijabli iz opisa aplikacije s oznakama u predlošcima postaje složeniji, iz slijedećih razloga:

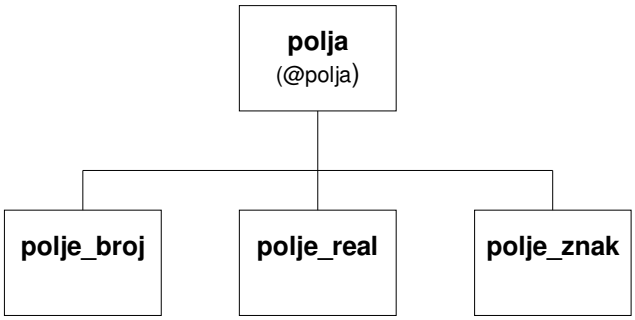
- moguće je više razina zamjenskih varijabli
- na svakoj razini moguće je 0..N pojava odgovarajuće zamjenske varijable

4.3.2.1 Skriptni model višerazinskog generatora

Višerazinski generator aplikacija također možemo modelirati pomoću odgovarajućih dijagrama skriptnog modela:

- dijagrama parametara opisa aplikacije i
- dijagrama metaskripata

Dijagram parametara opisa aplikacija za višerazinski generator aplikacija sadrži više razina (slika 4.12):



Slika 4.12: Dijagram parametara opisa višerazinskog generatora aplikacija iz primjera

Na slici NN može se vidjeti da će u opisu aplikacije postojati oznaka 'polja' koja uključuje oznake 'polje_broj', 'polje_real' i 'polje_znak' koja se odnose na tipove podataka koje ćemo koristiti. Prema tome, u opisu aplikacije potrebno je zadati podatke s kojima ta aplikacija radi i njihove tipove. Uz oznaku 'polja' u opisu aplikacije neće se navoditi vrijednost nego će se ona koristiti samo kao oznaka odjeljka za definiciju polja. Korištenje vrijednosti za pojedine oznake definirano je dijagramom metaskripata.

Odgovarajući opis aplikacije izgledao bi ovako:

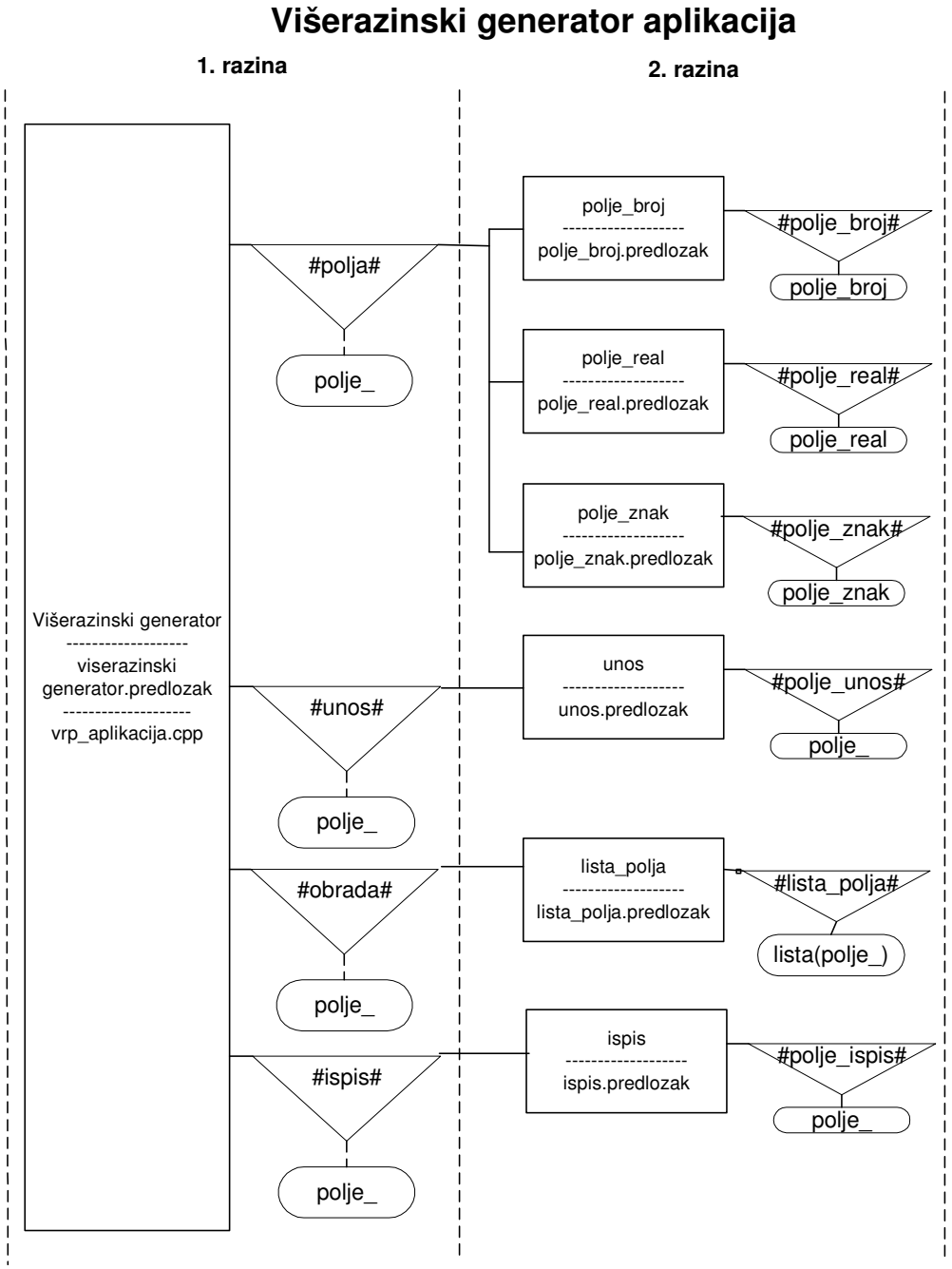
polja:
polje_<tip_polja>:<vrijednost>

pri čemu tip polja predstavlja jedna od tri ponuđene vrijednosti: broj, real i znak.

Sada možemo formirati primjer opisa aplikacije:

polja:
polje_broj:prva
polje_real:druga
polje_broj:treca

Oznake tipa bit će u aplikaciji zamijenjene tipovima iz odgovarajućih predložaka (prema programskom jeziku aplikacije), što se može vidjeti u dijagramu metaskripata (slika 4.13).



Slika 4.13: Dijagram metaskripata višerazinskog generatora aplikacija iz primjera

Na dijagramu možemo vidjeti distribuciju oznaka iz dijagrama parametara opisa aplikacije (navode se kao izvori) na odgovarajuće metaskripte. Kod predložaka 'unos', 'lista_polja' i 'ispis' primijenjena je racionalizacija, odnosno izvori su definirani s 'polje_', odnosno, odnose se na sve oznake koje u nazivu sadrže 'polje_'. Treba primijetiti da je ovdje korištena mogućnost predobrade za pojedine izvore, i to unutar izvora 'lista(polja_)'. To znači da funkcija lista treba obraditi sve izvore koji u nazivu sadrže 'polje_'. U konkretnom slučaju to je potrebno za stvaranje liste polja, gdje su elementi međusobno odvojeni zarezom, no iza zadnjeg elementa liste nema zareza. Nadalje, vidljiva je hijerarhija predložaka prema kojoj se generira aplikacija. Glavni predložak definira izlaznu datoteku u koju se upisuje generirani kod aplikacije ('vrp_aplikacija.cpp').

4.3.2.2 Primjer: razvoj jednostavnog generatora aplikacija u C++

U primjeru je prikazan razvoj jednostavnog generatora aplikacija u C++ koja definira podatke, omogućuje unos njihovih vrijednosti s tipkovnice, jednostavnu obradu i ispis. U odnosu na prethodni primjer, opis aplikacije je fleksibilniji, tako da je moguće koristiti proizvoljan broj varijabli različitih tipova.

Primjer takvog programa je slijedeći:

```
#include <iostream.h>
int prva;                                          deklaracije podataka
float druga;
char treca[40];
void main(){
//unos vrijednosti varijabli
cout << "prva = ";
cin >> prva;
cout << "druga = ";                                unos podataka
cin >> druga;
cout << "treca = ";
cin >> treca;
//obrada - formiranje i ispis liste polja
cout << "Lista polja:prva,druga,treca";            ispis liste polja
// ispis vrijednosti varijabli
cout << endl << "--------------------" << endl;
cout << "prva = ";
cout << prva << endl;
cout << "druga = ";                                ispis podataka
cout << druga << endl;
cout << "treca = ";
cout << treca << endl;
}
```

4.3.2.2.1 Izrada višerazinskog generatora prema skriptnom modelu

Skriptni model definira strukturu opisa aplikacije (dijagram parametara opisa aplikacije), distribuciju zadanih svojstava na pojedine metaskripte, te veze među metaskriptama.

Koraci rješavanja problema:

1. Utvrditi potrebne zamjenske varijable, njihove oznake i predloške (tablica 4.2):

zamjenska varijabla	oznaka	predložak
polja	#polja#	Višerazinski generator
polje_broj	# polje_broj #	polje_broj,unos,lista_polja,ispis
polje_real	# polje_real #	polje_real,unos,lista_polja,ispis
polje_znak	# polje_znak #	polje_znak,unos,lista_polja,ispis

Tablica 4.2: Zamjenske varijable i njihove oznake za višerazinski generator

2. Izraditi predloške aplikacije u ciljnom programskom jeziku (ovdje C++):

Višerazinski generator:

```
#include <iostream.h>
#polja#
void main(){
//unos vrijednosti varijabli
#unos#
//obrada - formiranje i ispis liste polja
#obrada#
// ispis vrijednosti varijabli
        cout << "--------------------" << endl;
#ispis#
}
```

polje_broj:

```
int #polje_broj#;
```

polje_real:

```
float #polje_real#;
```

polje_znak:

```
char #polje_znak#[40];
```

unos:

```
cout << "#polje_unos# = ";
cin >> #polje_unos#;
```

lista_polja:

```
cout << "Lista polja:#lista_polja#";
```

ispis:

```
cout << "#polje_ispis# = ";
cout << #polje_ispis# << endl;
```

3. Definirati vrijednosti zamjenskih varijabli (opis aplikacije):

```
polja:
polje_broj:prva
polje_real:druga
polje_znak:treca
```

4. Izraditi odgovarajući generator u jeziku skripata (ovdje Perl):

```
# Učitavanje glavnog predloška aplikacije
open (dat,"<višerazinski generator.predlozak");
@predlozak=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<polje_broj.predlozak");
@polje_broj=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<polje_real.predlozak");
```

učitavanje predložaka koda

```
@polje_real=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<polje_znak.predlozak");
@polje_znak=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<unos.predlozak");
@unos=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<polje_ispis.predlozak");
@polje_ispis=<dat>;
close (dat);
# Učitavanje predloška
open (dat,"<lista_polja.predlozak");
@lista_polja=<dat>;
close (dat);
```

(metaskripata)

```
open (dat,"<opis višerazinske aplikacije.txt");
@opis=<dat>;
close (dat);
```

Učitavanje opisa višerazinske aplikacije

```
#----------------------ZAJEDNIČKI POTPROGRAMI--------------------------
```

```
sub lista{
@p = @_;
$lista="";
foreach $line(@p){
($a,$b)=split(/:/,$line);
$lista="$lista$b,";
}#foreach
$lista=substr($lista,0,length($lista)-1);
return $lista;
}#lista
```

primjer predobrade izvora (stvaranje liste)

```
#-----------------------------------------------POTPROGRAMI-----------------------------------------
#----------------------ZAMJENE OZNAKA ZAMJENSKIM VRIJEDNOSTIMA-----------------
```

```
sub polja{
$zamjene="";
@p=@polja;
foreach $line(@p){
$redak=$line;
if ($redak =~ /polje_broj:/){
$l=$redak;
$l =~ s/polje_broj://;
@po=@polje_broj;
foreach $line (@po){
$line=~ s/#polje_broj#/$l/;
$zamjene="$zamjene$line";}}
```

#polje_broj#

```
if ($redak =~ /polje_real:/){
$l=$redak;
$l =~ s/polje_real://;
@po=@polje_real;
foreach $line (@po){
$line=~ s/#polje_real#/$l/;
$zamjene="$zamjene$line";}}
```

#polje_real#

```
if ($redak =~ /polje_znak:/){
$l=$redak;
$l =~ s/polje_znak://;
@po=@polje_znak;
foreach $line (@po){
$line=~ s/#polje_znak#/$l/;                    #polje_znak#
$zamjene="$zamjene$line";}}
}#foreach
$polja=$zamjene;
}#polja
```

```
sub unos{
$unos="";
@p=@polja;
foreach $line(@p){
if ($line =~ /polje_broj:/){
$line =~ s/polje_broj://;}                     #unos#
if ($line =~ /polje_real:/){
$line =~ s/polje_real://;}
if ($line =~ /polje_znak:/){
$line =~ s/polje_znak://;}
$l=$line;
@u=@unos;
foreach $line(@u){
$line =~ s/#polje_unos#/$l/;                   #polje_unos#
$unos="$unos$line";
}#foreach
}#foreach

}#unos
```

```
sub obrada{
$obrada="";
@pom_=@lista_polja;
$rez_=&lista(@polja);
foreach $line(@pom_){                          #lista_polja#
$line=~ s/#lista_polja#/$rez_/;
$obrada="$obrada$line";
}#foreach
}#obrada
```

```
sub ispis{
$zamjene="";
@p=@polja;
foreach $line(@polja){
($a,$polje)=split(/:/,$line);
@pom_=@polje_ispis;
foreach $line(@pom_){
$line=~ s/#polje_ispis#/$polje/;                #polje_ispis#
$zamjene="$zamjene$line";
}#foreach

}#foreach
$ispis=$zamjene;
}#ispis
```

```
#----------------------KRAJ POTPROGRAMA--------------------------------
```

```
#Čitanje vrijednosti zamjenskih varijabli iz opisa aplikacije
```

```
@o=@opis;
$faza_obrade=""; #obrada se dijeli na faze - svaka zamjenska varijabla prve razine predstavlja novu fazu
@polja=();
foreach $line(@o){ #petlja za čitanje opisa aplikacije
$line=~ s/\n//; #uklanjanje znaka za kraj retka
if ($line eq "polja:"){$faza_obrade="polja";}
if ($faza_obrade eq "polja"){          učitavanje zamjenskih vrijednosti
if ($line =~ /polje_broj/){            za oznake
push (@polja,$line);}
if ($line =~ /polje_real/){
push (@polja,$line);}
if ($line =~ /polje_znak/){
push (@polja,$line);}
} #kraj - faza_obrade=polja
} # kraj petlje za čitanje opisa aplikacije
```

#Za oznake u glavnom predlošku koji se zamjenjuju novim predlošcima kreiraju se pozivi potprograma

```
&polja;
&unos;                                 pozivi potprograma
&obrada;                               (2. razina dijagrama metaskripata)
&ispis;
```

#Pridruživanje vrijednosti zamjenskih varijabli oznakama u glavnom predlošku aplikacije i kreiranje ciljne aplikacije

```
open (dat,">vr_aplikacija.cpp");
@p=@predlozak;
foreach $line(@p){ #petlja za čitanje predloška aplikacije
#zamjene oznaka vrijednostima zamjenskih varijabli
$line=~ s/#polja#/$polja/;
$line=~ s/#unos#/$unos/;               zamjena oznaka u glavnom predlošku
$line=~ s/#obrada#/$obrada/;           (1. razina dijagrama metaskripata) i
$line=~ s/#ispis#/$ispis/;             kreiranje izlazne programske datoteke
#upis programskog retka u ciljnu aplikaciju
print dat $line;
} # kraj petlje za čitanje predloška aplikacije
close (dat);
```

print "vr_aplikacija.cpp\n";

4.3.3 Parametrizirani višerazinski generator

Parametrizirani višerazinski generator poopćava proces generiranja aplikacija, tako što omogućuje zadavanje pojedinih operacija generiranja pomoću poziva funkcija, s parametrima koji predstavljaju elemente skriptnog modela generatora. Da bi se takav generator mogao izraditi, trebalo je najprije izdvojiti funkcije generiranja, kao standardizirane postupke za sve generatore prema skriptnom modelu.

4.3.3.1 Funkcije generiranja prema skriptnom modelu

Radi automatizacije izrade generatora aplikacija prema skriptnom modelu potrebno je izdvojiti određene standardne postupke generiranja koji su zajednički za sve takve

generatore. To je realizirano pomoću funkcija generiranja, čime je omogućeno zadavanje procesa generiranja pomoću parametara.

Struktura generatora određena je dijagramom metaskripata. Dijagram metaskripata (slika 4.13) definira predloške programskog koda, izvore i njihove međusobne veze, pri čemu se postižu slijedeća svojstva:

- kod jednostavnog generatora oznake u predlošcima direktno se zamjenjuju izvorima podataka,
- višerazinski generator dobije se superpozicijom više jednostavnih jednorazinskih generatora.

Postupak izrade takvog višerazinskog parametriziranog generatora sastoji se od slijedećih koraka:

1. pomoću dijagrama parametara opisa generatora definira se struktura opisa aplikacije, koji generator koristi za generiranje aplikacija
2. pomoću dijagrama metaskripata odrede se veze između predložaka koda i njihovih izvora podataka.
3. izradi se parametrizirani višerazinski generator, koji uključuje slijedeće postupke, odnosno funkcije za njihovu realizaciju:

- Učitavanje predložaka koda, u odgovarajuća polja, npr.:

 # Učitavanje glavnog predloška aplikacije
@predlozak=&ucitaj_predlozak("višerazinski generator.predlozak");

- Čitanje opisa aplikacije, vrši se prema dijagramu parametara opisa aplikacije, od zadane početne do završne zamjenske varijable (ili do kraja opisa) vrši se pomoću odgovarajuće funkcije (ovdje &citaj_opis):

varijabla/polje=&citaj_opis(polje_s_opisom,od,do);

ili u primjeru:

@polja=&citaj_opis(@opis,"|","polja:","<kraj>");

- Generiranje zamjenskih vrijednosti za pojedine oznake u predlošcima vrši se prema dijagramu metaskripata, odnosno, za oznake u glavnom predlošku koji se zamjenjuju novim predlošcima kreiraju se pozivi potprograma.
Višerazinski generator dobije se superpozicijom više jednostavnih jednorazinskih generatora, pa se zbog toga najprije generiraju pojedine grane iz dijagrama metaskripata, čime se višerazinski generator u konačnici svodi na jednorazinski:

a) generiranje vrijednosti pojedinih grana:

varijabla=generiraj(@podaci,"|","tip:&zamjena");

ili u primjeru:

$polja=&generiraj(@polja,"|","polje_broj:&zamjena(\@polje_broj,'|','#polje_broj#',\$v)","polje_real:&zamjena(\@polje_real,'|','#polje_real#',\$v)","polje_znak:&zamjena(\@polje_znak,'|','#polje_znak#',\$v)");

Ovdje se aktiviraju tri jednostavna generatora iz dijagrama metaskripata (slika 4.13).

U nekim slučajevima potrebna je **predobrada ulaznih parametara**, kao što je u slijedećem primjeru formiranje liste ulaznih parametara. To se rješava odgovarajućim potprogramima, npr.:

```
#koristi se potprogram &lista za formiranje liste polja u obliku polje1,polje2, ... , poljeN
$l=&lista(@polja);
$obrada=&generiraj("$l:$l","|","$l:&zamjena(\@lista_polja,'|','#lista_polja#',\$v)");
```

i

b) generiranje aplikacije (kao kod jednorazinskog generatora). Vrijednosti pojedinih oznaka u glavnom predlošku aplikacije zamjenjuju se vrijednostima pojedinih varijabli dobivenih u prethodnom koraku:

```
#polje=zamjena(polje,oznaka,vrijednost uznake);

@predlozak=&zamjena(@predlozak,"|","#polja#",$polja);
@predlozak=&zamjena(@predlozak,"|","#unos#",$unos);
@predlozak=&zamjena(@predlozak,"|","#obrada#",$obrada);
@predlozak=&zamjena(@predlozak,"|","#ispis#",$ispis);
```

Osim spomenutih postupaka koristi se još:

- **Spremanje generirane aplikacije** u odgovarajuću programsku datoteku.

Na dijagramu metaskripata (slika 4.13) vidi se da pojedine metaskripte imaju definiran izlaz, obično u obliku izlazne programske datoteke. U primjeru polje @predlozak sadrži kod cijele aplikacije, pa ga možemo spremiti u odgovarajuću programsku datoteku:

```
&spremi(@predlozak,"|","vrp_aplikacija.cpp");
```

4.3.3.2 Gradivni elementi višerazinskog parametriziranog generatora

U razvijenom primjeru višerazinskog parametriziranog generatora možemo vidjeti implementaciju funkcija generiranja, te strukturu generatora, uz upotrebu odgovarajućih funkcijskih poziva.

Programski kod višerazinskog parametriziranog generatora:

```
$pamti=0; #pozicija kod čitanja opisa datoteke
# Učitavanje glavnog predloška aplikacije
@predlozak=&ucitaj_predlozak("višerazinski generator.predlozak");
# Učitavanje ostalih predložaka
@polje_broj=&ucitaj_predlozak("polje_broj.predlozak");
@polje_real=&ucitaj_predlozak("polje_real.predlozak");
@polje_znak=&ucitaj_predlozak("polje_znak.predlozak");
@unos=&ucitaj_predlozak("unos.predlozak");
@polje_ispis=&ucitaj_predlozak("polje_ispis.predlozak");
@lista_polja=&ucitaj_predlozak("lista_polja.predlozak");
# Učitavanje opisa višerazinske aplikacije
@opis=&ucitaj_predlozak("opis višerazinske aplikacije.txt");
```

učitavanje predložaka u polja s neograničenim brojem elemenata

```
#----------------------ZAJEDNIČKI POTPROGRAMI--------------------------
```

```
sub ucitaj_predlozak{
local @predlozak,$naziv_dat;
$naziv_dat=@_[0];
open (dat,"<$naziv_dat");
@predlozak=<dat>;
close (dat);
return @predlozak;
}#ucitaj_predlozak
```

funkcija za učitavanje predložaka

```
sub citaj_opis{
local @ulaz,@opis,$sprema,@podaci;
#čitanje parametara @opis,"|",$od,$do
@ulaz=@_;
@opis=();
$sprema="0";
$brojac=0;
foreach $line(@ulaz){
$brojac++;
if (($line ne "\n")&&($brojac>=$pamti)){
$pamti=$brojac;
if ($line eq "|"){$sprema="1";}
if (($line ne "|")&&($sprema eq "2")){$sprema="3";$do=$line;}
if (($line ne "|")&&($sprema eq "1")){$sprema="2";$od=$line;}
if ($sprema eq "0"){
push (@opis,$line);}
}#if
}#foreach
@podaci=();
$sprema="0";
foreach $line(@opis){ #petlja za čitanje opisa aplikacije
$line=~ s/\n//; #uklanjanje znaka za kraj retka
if ($line =~ /$do/){$sprema="0";}
if ($sprema eq "1"){
push (@podaci,$line);}
if ($line =~ /$od/){$sprema="1";}
} # kraj petlje za čitanje opisa aplikacije
return @podaci;
}#citaj opis
```

funkcija za čitanje opisa aplikacije

```
sub zamjena{
local @pred,$sprema,@zamjena,$tag,$vrijednost,@izlaz;
#@predlozak=zamjena(@predlozak,",","",#polja#",$polja);
#čitanje parametara @predlozak,"|",$tag,$vrijednost
@pred=@_;
@zamjena=();
$sprema="0";
foreach $line(@pred){
if ($line eq "|"){$sprema="1";}
if (($line ne "|")&&($sprema eq "2")){$sprema="3";$vrijednost=$line;}
if (($line ne "|")&&($sprema eq "1")){$sprema="2";$tag=$line;}
if ($sprema eq "0"){
push (@zamjena,$line);}
}#foreach
@izlaz=();
foreach $line(@zamjena){
$line=~ s/$tag/$vrijednost/;
$line=~ s/$tag/$vrijednost/;
$line=~ s/$tag/$vrijednost/;
push (@izlaz,$line);}
return @izlaz;
```

funkcija za zamjenu oznaka zamjenskim vrijednostima

```
}#zamjena
```

```
sub spremi{
local @predlozak,$sprema,$spremi,$dat;
@predlozak=@_;
#&spremi(@predlozak,"|","vrp_aplikacija.cpp");
$sprema="0";
$spremi="";
foreach $line(@predlozak){
if ($line eq "|"){$sprema="1";}
if (($line ne "|")&&($sprema eq "1")){$sprema="2";$dat=$line;}
if ($sprema eq "0"){
$spremi="$spremi$line";}
}#foreach
open (dat,">$dat");
print dat $spremi;
close dat;
}#spremi
```

funkcija za spremanje generiranog koda u datoteku

```
sub generiraj{
local @param,$sprema,@podaci,@parovi,$rez,@p,$t1,$v,$tip,$poziv,@zam;
#varijabla=generiraj(@podaci,"|","tip:&zamjena"); //parovi
@param=@_;
$sprema="0";
@podaci=();
@parovi=();
$rez="";
foreach $line(@param){
if ($line eq "|"){$sprema="1";}
if ($sprema eq "0"){
push (@podaci,$line);}
if (($sprema eq "1")&&($line ne "|")){
push (@parovi,$line);}
}#foreach
foreach $line (@podaci){
#print ">$line\n";
($t1,$v)=split(/:/,$line);
@p=@parovi;
foreach $line (@p){
($tip,$poziv)=split(/:/,$line);
if ($t1 =~ /$tip/){
@zam=eval "$poziv";
foreach $line (@zam){
$rez="$rez$line";
}#foreach
}#if
}#foreach
}#foreach
return $rez;
}#generiraj
```

funkcija za generiranje pojedine grane dijagrama metaskripata

```
#----------------------MODIFIKACIJE ULAZNIH PODATAKA-------------------------------
```

```
sub lista{
local @p,$lista,$a,$b;
@p = @_;
$lista="";
foreach $line(@p){
($a,$b)=split(/:/,$line);
$lista="$lista$b,";
}#foreach
```

primjer funkcije za predobradu ulaznih parametara

```
$lista=substr($lista,0,length($lista)-1);
return $lista;
}#lista
#----------------------KRAJ ZAJEDNIČKIH POTPROGRAMA-------------------------------

#Čitanje vrijednosti zamjenskih varijabli iz opisa aplikacije
#varijabla/hash=&citaj_opis(hash_s_opisom,od,do);
@polja=&citaj_opis(@opis,"|","polja:","<kraj>");

#Za oznake u glavnom predlošku koji se zamjenjuju novim predlošcima kreiraju se pozivi potprograma
#varijabla=generiraj(@podaci,"|","tip:&zamjena");

$polja=&generiraj(@polja,"|","polje_broj:&zamjena(\@polje_broj,'|','#polje_broj#',\$v)",
"polje_real:&zamjena(\@polje_real,'|','#polje_real#',\$v)","polje_znak:&zamjena(\@pol
je_znak,'|','#polje_znak#',\$v)");
#&polja;
$unos=&generiraj(@polja,"|","polje_:&zamjena(\@unos,'|','#polje_unos#',\$v)");
#&unos;
$l=&lista(@polja);
$obrada=&generiraj("$l:$l","|","$l:&zamjena(\@lista_polja,'|','#lista_polja#',\$v)");
#&obrada;
$ispis=&generiraj(@polja,"|","polje_:&zamjena(\@polje_ispis,'|','#polje_ispis#',\$v)");
#&ispis;

#Pridruživanje vrijednosti zamjenskih varijabli oznakama u glavnom predlošku aplikacije i kreiranje ciljne aplikacije
#polje=zamjena(polje,oznaka,vrijednost oznake);
@predlozak=&zamjena(@predlozak,"|","#polja#",$polja);
@predlozak=&zamjena(@predlozak,"|","#unos#",$unos);
@predlozak=&zamjena(@predlozak,"|","#obrada#",$obrada);
@predlozak=&zamjena(@predlozak,"|","#ispis#",$ispis);
&spremi(@predlozak,"|","vrp_aplikacija.cpp");
print "vrp_aplikacija.cpp\n";
```

implementacija dijagrama metaskripata kroz funkcijske pozive i spremanje generiranog koda

4.3.3.3 Svojstva višerazinskog parametriziranog generatora

Realizacijom parametriziranog višerazinskog generatora ostvareno je slijedeće:

1. Ostvareno je osnovno svojstvo neophodno za izradu generatora prema zadanom skriptnom modelu, a to je da se višerazinski generator dobije superpozicijom više jednostavnih jednorazinskih generatora.
2. Izdvojene su pojedine zajedničke funkcije generatora, pozivi kojih se formiraju prema odgovarajućim dijagramima iz skriptnog modela:

- funkcija za učitavanje predložaka koji se koriste u generiranju
- funkcija za učitavanje opisa aplikacije. Njeni pozivi definiraju se u skladu s dijagramom parametara opisa generatora.
- funkcija za generiranje pojedine grane iz dijagrama metaskripata, čime se višerazinski generator u konačnici svodi na jednorazinski.
- funkcija za direktnu zamjenu vrijednosti oznaka iz glavnog predloška aplikacije vrijednostima pojedinih grana.
- funkcije za predobradu vrijednosti zamjenskih varijabli (npr. za formiranje raznih lista)
- funkcija za spremanje generirane aplikacije u odgovarajuću programsku datoteku.

3. Uočene su prednosti primjene jezika skripata, kao što je Perl, za realizaciju generatora aplikacija:

- koristi se mogućnost evaluacije znakovnih nizova (funkcija eval u Perl-u),
- koriste se mogućnosti jezika Perl u obradi znakovnih nizova,
- koriste se polja neograničene veličine i
- koriste se potprogrami s neograničenim brojem parametara.

4. Višerazinski parametrizirani generator predstavlja osnovu za realizaciju metageneratora, koji će direktno transformirati zadani skriptni model generatora u odgovarajući generator aplikacija (ili komponenta).

4.3.4 Automatizacija izrade generatora

Nakon odvajanja programskog koda potprograma koji su zajednički za sve generatore dobivamo programski kod, koji je definiran s dva dijagrama skriptnog modela (slika 4.14). Dio koda označen s 'A' definiran je dijagramom parametara opisa aplikacije (definira izvore podataka), a dio označen s 'B' definiran je dijagramom metaskripata i sadrži funkcijske pozive za generiranje.

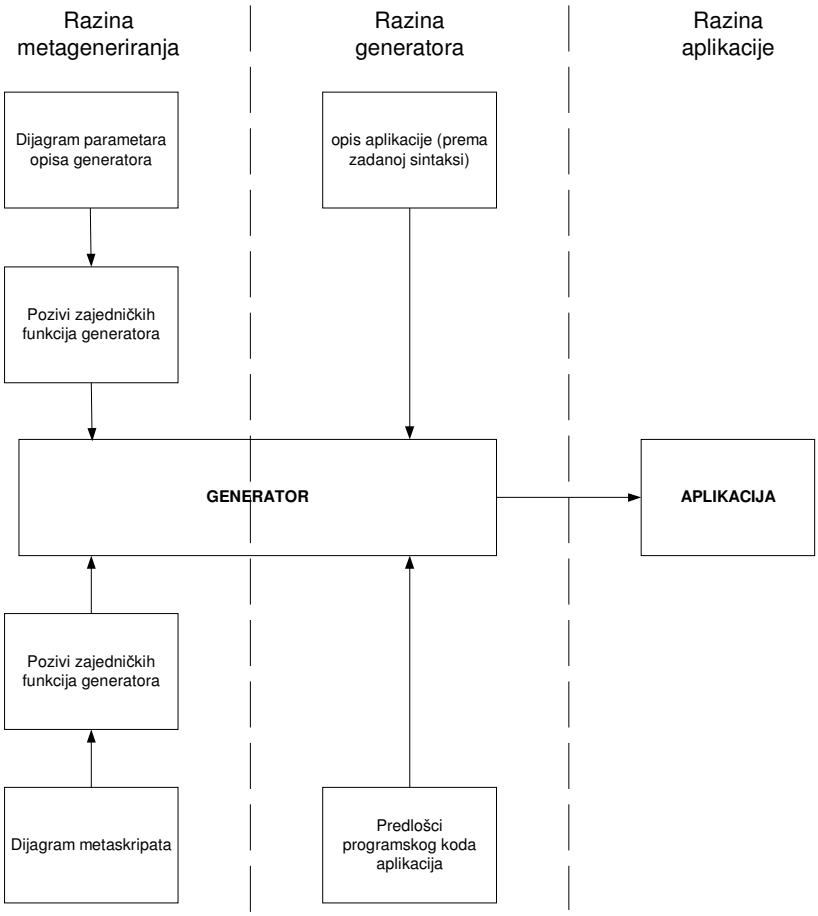
A

```
#Primjer generatora
require ("potprogrami.pm");
@predlozak=&ucitaj_predlozak("višerazinski generator.predlozak");
@polje_broj=&ucitaj_predlozak("polje_broj.predlozak");
@polje_real=&ucitaj_predlozak("polje_real.predlozak");
@polje_znak=&ucitaj_predlozak("polje_znak.predlozak");
@unos=&ucitaj_predlozak("unos.predlozak");
@polje_ispis=&ucitaj_predlozak("polje_ispis.predlozak");
@lista_polja=&ucitaj_predlozak("lista_polja.predlozak");
@opis=&ucitaj_predlozak("opis višerazinske aplikacije.txt");
@polja=&citaj_opis(@opis,"|","polja:","<kraj>");
$polja=&generiraj(@polja,"|","polje_broj:&zamjena(\@polje_broj,'|','#polje_broj#',\$v)","polje_r
eal:&zamjena(\@polje_real,'|','#polje_real#',\$v)","polje_znak:&zamjena(\@polje_znak,'|','#polj
e_znak#',\$v)");
$unos=&generiraj(@polja,"|","polje_:&zamjena(\@unos,'|','#polje_unos#',\$v)");
$l=&lista(@polja);
$obrada=&generiraj("$l:$l","|","$l:&zamjena(\@lista_polja,'|','#lista_polja#',\$v)");
$ispis=&generiraj(@polja,"|","polje_:&zamjena(\@polje_ispis,'|','#polje_ispis#',\$v)");
@predlozak=&zamjena(@predlozak,"|","#polja#",$polja);
@predlozak=&zamjena(@predlozak,"|","#unos#",$unos);
@predlozak=&zamjena(@predlozak,"|","#obrada#",$obrada);
@predlozak=&zamjena(@predlozak,"|","#ispis#",$ispis);
&spremi(@predlozak,"|","vrp_aplikacija.cpp");
print "vrp_aplikacija.cpp\n";
```

B

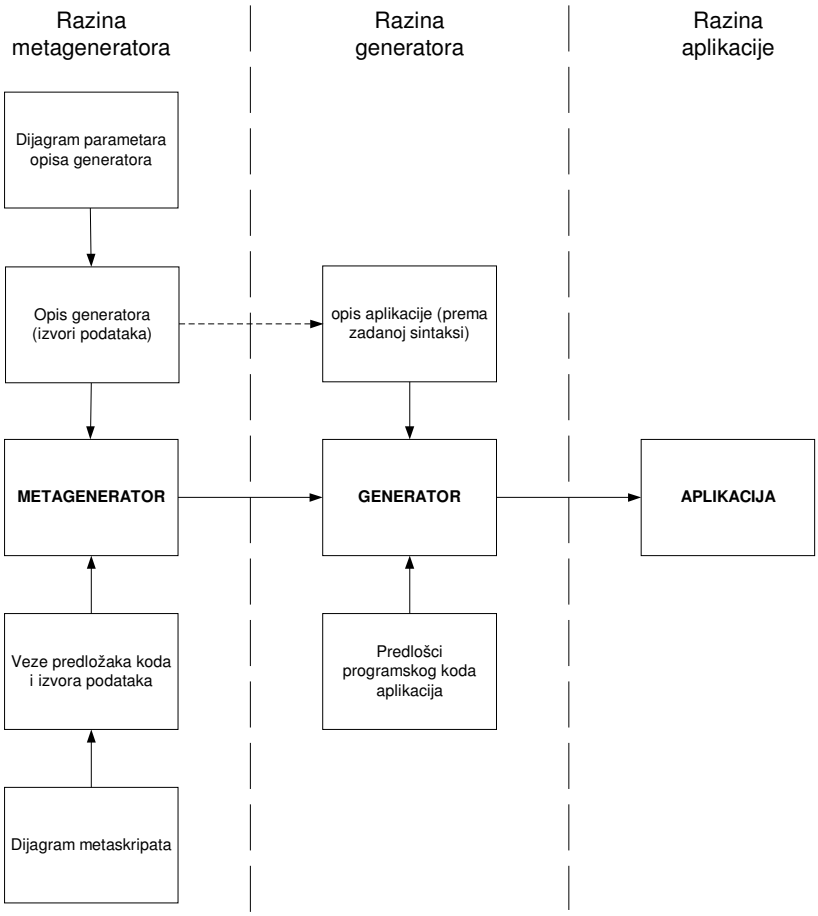
Slika 4.14: Programski kod generatora nakon odvajanja zajedničkih potprograma

Takva struktura sustava za generiranje aplikacija vidljiva je na slici 4.15. U odnosu na strukturu sustava uz upotrebu metageneratora (slika 4.16) izbjegnuta je jedna razina generiranja.



Slika 4.15: Sustav za generiranje aplikacija (1)

U slučaju metageneratora, postojala bi još jedna razina generiranja, odnosno, programski kod sa slike 4.15 dobio bi se generiranjem iz odgovarajućeg opisa, kao što pokazuje slika 4.16:



Slika 4.16: Sustav za generiranje aplikacija (2)

Dakle, u slučaju metageneratora uvela bi se po jedna međurazina u obliku tekstualnog opisa, između dijagrama parametara opisa generatora i metageneratora, dijagrama metaskripata i metageneratora.

4.3.5 Metagenerator

Metagenerator je program koji, prema zadanom skriptnom modelu, generira programski kod generatora. Za generiranje aplikacije koristi se opis koji predstavlja tekstualni oblik skriptnog modela. Takav opis sastoji se od odjeljaka i definicija unutar odgovarajućih odjeljaka.

4.3.5.1 Odjeljak za učitavanje predložaka (metaskripata) koje će koristiti generator:

__PREDLOSCI__

- pridružuje poljima s neograničenim brojem elemenata izvore predložaka koda (npr. odgovarajuće nazive tekstualnih datoteka).
Pojedini predložak definira se na slijedeći način:

@<naziv_polja>=<izvor>

Primjer:

@predlozak=višerazinski generator.predlozak

Ovdje je sadržaj polja 'predlozak' definiran sadržajem datoteke 'višerazinski generator.predlozak'.

4.3.5.2 Odjeljak za učitavanje opisa aplikacije koji će koristiti generator:

__OPIS__

- pridružuje polju s neograničenim brojem elemenata izvor koji sadrži opis aplikacije, na temelju kojeg generator generira aplikaciju.

Sintaksa odgovara sintaksi u prethodnom odjeljku za učitavanje predložaka:

@<naziv_polja>=>izvor>

Primjer:

@opis=opis višerazinske aplikacije.txt

4.3.5.3 Odjeljak za učitavanje opisa aplikacije

__CITANJE OPISA__

U odjeljku se definira oznaka iz opisa aplikacije od koje započinje čitanje opisa, polje s neograničenim brojem elemenata koje će sadrži opis aplikacije i poziciju u opisu do koje se opis čita (postoji mogućnost da isti opis definira više od jedne aplikacije).

Čitanje opisa zadaje se prema slijedećoj sintaksi:

<oznaka iz opisa aplikacije>,<izvor opisa aplikacije>,<oznaka kraja čitanja>

Primjer:

polja:,@polja,<kraj>

U primjeru se definira čitanje opisa aplikacije, koji se nalazi u polju '@polja' od oznake 'polja:' do kraja opisa (oznaka <kraj> ne mora se nalaziti u opisu, nego predstavlja kraj opisa).

4.3.5.4 Odjeljak za generiranje aplikacije

__GENERIRANJE__

Ovaj odjeljak predstavlja tekstualni oblik dijagrama metaskripata, jer svaki njegov redak definira jednu njegovu granu. Sintaksa pojedinog retka je slijedeća:

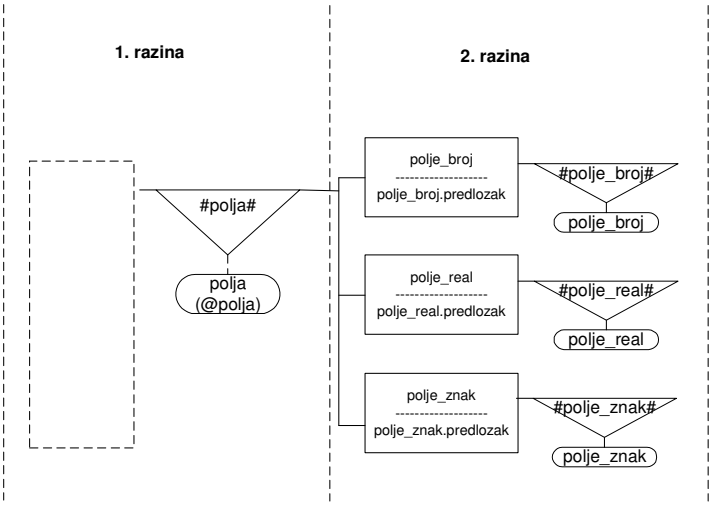
<predlozak>=<izvor>,<oznaka_za_vezu>|<definicija grane niže razine>|<definicija grane niže razine>|<definicija grane niže razine>| .....

Definicija grane niže razine izgleda ovako:

|<oznaka_iz_opisa_aplikacije>,<predlozak>,<oznaka za vezu>

Primjer:

Dijagram metaskripata (slika 4.17):



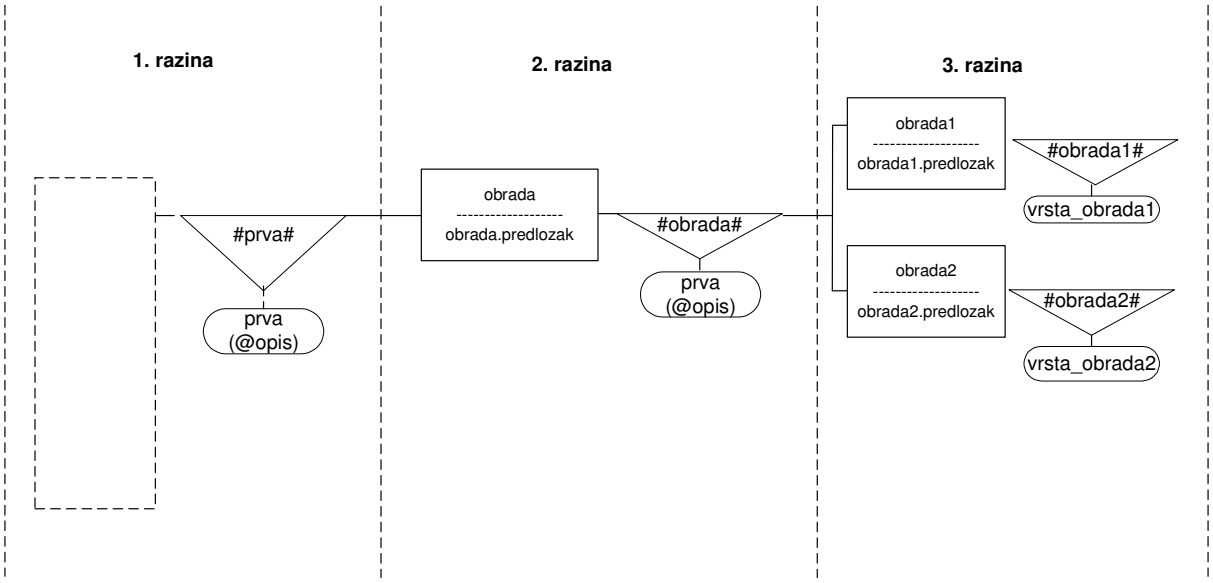
Slika 4.17: Primjer dijagrama metaskripata za jednu izdvojenu granu

Odgovarajući tekstualni opis izgleda ovako:

@predlozak=@polja,#polja#|polje_broj,@polje_broj,#polje_broj#|polje_real, @polje_real,#polje_real#|polje_znak,@polje_znak,#polje_znak#

Ovdje je definirano generiranje zamjenske vrijednosti za oznaku '#polja#', koja se nalazi unutar predloška '@predlozak'.
U slučaju da grane niže razine sadrže daljnje podgrane, potrebno je u prethodnom retku definirati generiranje podgrane, a zatim je uvrstiti u granu više razine.

Primjer: Za slijedeći dijagram metaskripata (slika 4.18) potrebno je izraditi odgovarajući tekstualni opis:



Slika 4.18: Dijagram metaskripata s tri razine

Odgovarajući tekstualni opis izgleda ovako:

@obrada=@opis,#obrada#|vrsta_obrada1,@obrada1,#obrada1#|
vrsta_obrada2,@obrada2,#obrada2#

@predlozak=@opis,#prva#|prva,@obrada,#prva#

Najprije se definira generiranje grane 'obrada', a nakon toga i generiranje glavne grane.

4.3.5.5 Odjeljak za spremanje aplikacije

__SPREMANJE__

U odjeljku za spremanje aplikacije spremaju se polja s neograničenim brojem elemenata (koja sadrže generirani programski kod) u odgovarajuće izlazne datoteke.

<polje s generiranim kodom>=<naziv datoteke>

Primjer:

@predlozak=vrp_aplikacija.cpp

4.3.5.6 Korištenje komentara

Komentar se može koristiti u svim odjeljcima, a označava se s dvije kose crte '//':

//tekst komentara

4.3.6 Primjer: razvoj jednostavnog generatora aplikacija uz pomoć metageneratora

Jednostavan generator aplikacija u C++ iz poglavlja 4.3.2. sada možemo realizirati uz pomoć metageneratora. Kao što možemo vidjeti na slici 4.1, metagenerator koristi tekstualni oblik skriptnog modela generatora kao specifikaciju na temelju koje se izrađuje generator. Skriptni model generatora dat je na slikama 4.12. i 4.13.

4.3.6.1 Tekstualni opis generatora

Tekstualni opis generatora sadrži hijerarhiju elemenata dijagrama metaskripata zadanu u tekstualnom obliku (odjeljak '__GENERIRANJE__'), zajedno s pomoćnim operacijama, neophodnim za rad generatora:

- učitavanje predložaka koda iz predviđenih izvora (odjeljak '__PREDLOSCI__')
- učitavanje tekstualnog opisa aplikacije (odjeljak '__OPIS__')
- postepeno čitanje tekstualnog opisa aplikacije (odjeljak '__CITANJE OPISA__')
- spremanje generiranog programskog koda u odgovarajuću programsku datoteku (odjeljak '__SPREMANJE__')

Za primjer koji se obrađuje u ovom radu odgovarajući tekstualni opis generatora izgleda ovako:

```
//Opis metageneratora: Jednostavan generator aplikacija u C++

__PREDLOSCI__
@predlozak=višerazinski generator.predlozak
@polje_broj=polje_broj.predlozak
@polje_real=polje_real.predlozak
@polje_znak=polje_znak.predlozak
@unos=unos.predlozak
@polje_ispis=polje_ispis.predlozak
@lista_polja=lista_polja.predlozak
__OPIS__
@opis=opis višerazinske aplikacije.txt

__CITANJE OPISA__
//oznaka,izvor,kraj oznake
polja:,@polja,<kraj>

__GENERIRANJE__
//@predlozak=izvor,zam|oznaka,predlozak,zam|oznaka,predlozak,zam|
//viša razina:
//@polje_broj=...
@predlozak=@polja,#polja#|polje_broj,@polje_broj,#polje_broj#|polje_real,@polje_real,#polje_real#|polje_znak,@polje_znak,#polje_znak#
@predlozak=@polja,#unos#|polje_,@unos,#polje_unos#
@predlozak=&lista(@polja),#obrada#|lista_polja,@lista_polja,#lista_polja#
@predlozak=@polja,#ispis#|polje_,@polje_ispis,#polje_ispis#

__SPREMANJE__
//@predlozak=<naziv datoteke>
@predlozak=vrp_aplikacija.cpp
```

Odjeljak '__PREDLOSCI__' definira učitavanje svih predložaka koda (metaskripata) iz dijagrama metaskripata u odgovarajuća polja s neograničenim brojem elemenata, koja će se koristiti generator. Opis aplikacije nalazit će se u datoteci ' opis višerazinske aplikacije.txt ', odnosno, u polju '@opis'. Odjeljak '__CITANJE OPISA__' definira čitanje opisa aplikacije od oznake 'polja:' do kraja opisa. Četiri osnovne grane dijagrama metaskripata predstavljene su sa četiri retka opisa unutar odjeljka '__GENERIRANJE__', dok se u odjeljku '__SPREMANJE__' određuje spremanje glavnog predloška aplikacije (koji nakon generiranja definiranog u prethodnom odjeljku sadrži gotov programski kod aplikacije) u datoteku 'vrp_aplikacija.cpp'.

4.3.6.2 Izrađeni generator

Na temelju tekstualnog opisa iz prethodnog poglavlja metagenerator generira odgovarajući generator (slika 4.1).
Za primjer koji se obrađuje u ovom radu programski kod generatora u jeziku Perl izgleda ovako:

```
#Generator
require ("potprogrami.pm");
#Opis metageneratora: Jednostavan generator aplikacija u C++

#predlosci
@predlozak=&ucitaj_predlozak("višerazinski generator.predlozak");
@polje_broj=&ucitaj_predlozak("polje_broj.predlozak");
@polje_real=&ucitaj_predlozak("polje_real.predlozak");          učitavanje predložaka koda
@polje_znak=&ucitaj_predlozak("polje_znak.predlozak");
@unos=&ucitaj_predlozak("unos.predlozak");
@polje_ispis=&ucitaj_predlozak("polje_ispis.predlozak");
@lista_polja=&ucitaj_predlozak("lista_polja.predlozak");

#opis
@opis=&ucitaj_predlozak("opis višerazinske aplikacije.txt");
#citanje_opisa
#oznaka,izvor,kraj oznake                                       čitanje opisa aplikacije
@polja=&citaj_opis(@opis,'|','polja:','<kraj>');

#generiranje
#@predlozak=izvor,zam|oznaka,predlozak,zam|oznaka,predlozak,zam|

@predlozak=&zamjena(@predlozak,'|','#polja#',&generiraj(@polja,'|',"polje_broj:&zamjena(\@
polje_broj,'|','#polje_broj#',\$v)","polje_real:&zamjena(\@polje_real,'|','#polje_real#',\$v)","polje
_znak:&zamjena(\@polje_znak,'|','#polje_znak#',\$v)"));
@predlozak=&zamjena(@predlozak,'|','#unos#',&generiraj(@polja,'|',"polje_:&zamjena(\@uno
s,'|','#polje_unos#',\$v)"));
$l=&lista(@polja);
@predlozak=&zamjena(@predlozak,'|','#obrada#',&generiraj("$l:$l",'|',"$l:&zamjena(\@lista_p
olja,'|','#lista_polja#',\$v)"));
@predlozak=&zamjena(@predlozak,'|','#ispis#',&generiraj(@polja,'|',"polje_:&zamjena(\@polj
e_ispis,'|','#polje_ispis#',\$v)"));
#spremanje                                                      generiranje pojedinih grana
#@predlozak=<naziv datoteke>                                    prema dijagramu metaskripata

&spremi(@predlozak,'|','vrp_aplikacija.cpp');                   spremanje generiranog koda
```

Generator koristi biblioteku standardnih funkcija generiranja ('potprogrami.pm'), što bitno pojednostavljuje njegovu izradu, odnosno, ovdje generiranje.

4.3.6.3 Primjer generiranja aplikacije u C++

Generator za generiranje aplikacija koristi tekstualni opis aplikacije i odgovarajuće predloške programskog koda (metaskripte). Opis aplikacije definira specifična svojstva (aspekte) pojedine aplikacije unutar problemske domene, koja je određena predlošcima programskog koda (predlošci su zajednički za sve aplikacije unutar jedne problemske domene).

4.3.6.3.1 Opis aplikacije

Opis aplikacije u primjeru koji se obrađuje u ovom radu definira podatke s kojima aplikacija radi. Ovdje ćemo definirati da aplikacija koristi četiri varijable različitih tipova:

- varijabla cjelobrojna (cjelobrojnog tipa),
- varijabla realna (realnog tipa),
- varijabla niz (znakovni niz) i
- varijabla realni_broj (realnog tipa).

Odgovarajući opis aplikacije bio bi slijedeći:

polja:
polje_broj:cjelobrojna
polje_real:realna
polje_znak:niz
polje_real:realni_broj

4.3.6.3.2 Generirani kod aplikacije

Generirani kod aplikacije, prema prethodnom opisu, izgleda ovako:

```
#include <iostream.h>
int cjelobrojna;
float realna;                                   deklaracija varijabli
char niz[40];
float realni_broj;
void main(){
//unos vrijednosti varijabli
cout << "cjelobrojna = ";
cin >> cjelobrojna;
cout << "realna = ";                            učitavanje vrijednosti varijabli
cin >> realna;
cout << "niz = ";
cin >> niz;
cout << "realni_broj = ";
cin >> realni_broj;
//obrada - formiranje i ispis liste polja
cout << "Lista polja:cjelobrojna,realna,niz,realni_broj";     formiranje liste polja
// ispis vrijednosti varijabli
cout << endl << "--------------------" << endl;
```

```
cout << "cjelobrojna = ";
cout << cjelobrojna << endl;
cout << "realna = ";
cout << realna << endl;                    ispis vrijednosti varijabli
cout << "niz = ";
cout << niz << endl;
cout << "realni_broj = ";
cout << realni_broj << endl;
}
```

Usporedimo li ovaj programski kod s navedenim u poglavlju 4.3.2, možemo vidjeti da se definicija specifičnih svojstava programa može svesti na jednostavan opis koji se koristi za njegovo generiranje. Takva svojstva ne predstavljaju standardne programske jedinice kao što su potprogrami i klase, nego se mogu pojavljivati unutar različitih standardnih jedinica, odnosno, predstavljaju **aspekte**. Aspekti se distribuiraju na pojedine predloške programskog koda (metaskripte) prema dijagramu metaskripata.

4.4 Gramatika skriptnog modela

Gramatika skriptnog modela predstavlja formalizirani oblik njegove sintakse. Korištena standardna BNF notacija (eng. Backus-Naur Form) za definiranje sintakse grafičkog, odnosno tekstualnog oblika skriptnog modela. Sintaksa je relativno jednostavna i treba uočiti da za razliku od sintakse programskih jezika ne sadrži, osim u tekstualnoj specifikaciji generatora, preddefinirane ključne riječi.

Izrađene su slijedeće gramatike u BNF notaciji:

- gramatika tekstualnog opisa aplikacije,
- gramatika dijagrama parametara opisa aplikacije
- gramatika dijagrama metaskripata i
- gramatika tekstualnog opisa generatora (tekstualni oblik skriptnog modela koji koristi metagenerator)

Korištena su slijedeća pravila BNF-a (Yaxin, 1999., str 1-3.):

Meta-simboli BNF-a su slijedeći :

::= znači “definira se kao”
| znači “ili”
< > koriste se za navođenje naziva kategorija

Znakovi ‘<’ i ‘>’ omogućuju razlikovanje imena sintaksnih pravila (koja se također nazivaju i ne-terminalskim simbolima) od terminalskih simbola koji se pišu točno na način na koji trebaju biti reprezentirani. Pravilo BNF-a koje definira ne-terminal ima slijedeći oblik :

ne-terminal ::= slijed alternativa koje se sastoje od nizova znakova terminala ili ne-terminala odvojenih meta-simbolom |.

- opcijski elementi navode se unutar meta simbola **[** i **]**.
- ponavljajući elementi (nula ili više puta) uključeni su u meta simbole **{** i **}**.

- terminali od samo jednog znaka okruženi su navodnicima (") kako bi se razlikovali od meta simbola.

4.4.1 Gramatika tekstualnog opisa aplikacije

Tekstualni opis aplikacije predstavlja specifikaciju pojedine aplikacije unutar njene problemske domene, koju koristi odgovarajući generator za generiranje aplikacije.

Gramatika u BNF obliku izgleda ovako:

opis ::= red_opisa { red_opisa } EOF
red_opisa ::= oznaka ':' [vrijednost] EOLN
red_opisa_vise_razine ::= red_opisa { red_opisa } red_opisa_vise_razine | EOF
oznaka ::= <naziv_zajednickog_dijela> ['_' | '.' <naziv_specificnog_dijela>]
<naziv_specificnog_dijela> ::= <naziv_zajednickog_dijela_vise_razine> ['_' | '.'
    <naziv_specificnog_dijela_nize_razine>]
komentar ::= // znak { znak }

4.4.2 Gramatika dijagrama parametara opisa aplikacije

Dijagram parametara opisa aplikacije predstavlja hijerarhijski dijagram sa samo jednom vrstom elementa - oznakom. Zbog toga mu je i gramatika u BNF obliku vrlo jednostavna:

dijagram_parametara_opisa_aplikacije ::= oznaka { oznaka }
oznaka ::= <naziv_oznake> [izvor]
nadređena_oznaka :== [{oznaka}]

4.4.3 Gramatika dijagrama metaskripata

Dijagram metaskripata također je hijerarhijski dijagram, ali je nešto složeniji od dijagrama parametara opisa aplikacije jer se sastoji iz tri vrste elemenata, a to su metaskripta, veza i izvor. Gramatika u BNF obliku izgleda ovako:

dijagram_metaskripata::=element_dijagrama_metaskripata
    { element_dijagrama_metaskripata }
elementa_dijagrama_metaskripata ::= metaskripta, veza , izvor
metaskripta ::= <naziv_metaskripte> [//<komentar>] <izvor_koda> [
    izlaz_generiranog_koda]
veza ::= '#' < zamjenska_oznaka > '#'
oznaka ::= oznaka | <naziv_zajednickog_dijela> '_'
izvor ::= oznaka | '&'<naziv_funkcije> '(' oznaka ')'
jednorazinski_generator ::= metaskripta { veza, izvor }
viserazinski_generator ::= metaskripta { veza { jednorazinski_generator }, izvor }

4.4.4 Gramatika tekstualnog opisa generatora

Tekstualni opis generatora predstavlja tekstualni oblik skriptnog modela, prilagođen za automatiziranu izradu generatora pomoću metageneratora. Opis se sastoji od odjeljaka i definicija unutar tih odjeljaka. Odgovarajuća gramatika u BNF obliku je slijedeća:

```
tekstualni_opis ::= odjeljak { odjeljak } EOF
odjeljak ::= <naziv_odjeljka> EOLN <sadrzaj_odjeljka>
<naziv_odjeljka>  ::=  __PREDLOSCI__  |  __OPIS__  |  __CITANJE  OPISA__  |
        __GENERIRANJE__ | __SPREMANJE__
komentar ::= // znak { znak }
odjeljak_predlozaka ::= __PREDLOSCI__  EOLN { @<naziv_polja>=<izvor> EOLN }
odjeljak_opisa ::= __OPIS__  EOLN { @<naziv_polja>=<izvor> EOLN }
odjeljak_za_citanje_opisa ::= __CITANJE OPISA__  EOLN
        { specifikacija_citanja_opisa  EOLN }
specifikacija_citanja_opisa ::= <oznaka_iz_opisa_aplikacije> ','
        @<izvora_opisa_aplikacije>','<kraj>'
odjeljak_za_generiranje ::= __GENERIRANJE__  EOLN
        { pojedina_grana EOLN }
pojedina_grana ::= <predlozak>'='izvor,veza
        { '|'grana_nize_razine } EOLN
veza ::= '#' < zamjenska_oznaka > '#'
metaskripta ::= <naziv_metaskripte>
oznaka ::= oznaka | <naziv_zajednickog_dijela> '_'
izvor ::= oznaka | '&'<naziv_funkcije> '(' oznaka ')'
grana_nize_razine ::= oznaka ',' metaskripta ',' veza
odjeljak_spremanja ::= __SPREMANJE__  EOLN { @<naziv_polja>=
        <naziv_izlazne_datoteke> EOLN }
```

4.5 Odnos objektnog i skriptnog modela

Skriptni model generatora aplikacija razvijen je za potrebe modeliranja generatora aplikacija prema konceptu generativnog programiranja temeljenog na jezicima skripata. Sastoji se od dva dijagrama: dijagrama parametara opisa aplikacije i skriptnog modela. Specifičnosti ponuđenog skriptnog modela o odnosu na postojeći objektni model su slijedeće:

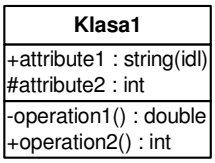
- definira generator aplikacija, a ne pojedinu aplikaciju (meta razina),
- generator je definiran kao struktura tipa stablo, pri ćemu se složeni višerazinski generator dobije superpozicijom jednostavnih jednorazinskih generatora,
- orijentiran je na definiranje zadanih, specifičnih aspekata budućih aplikacija unutar zadane problemske domene, a ne na sve funkcionalnosti aplikacija (jer se te definiraju na nižoj razini apstrakcije, u predlošcima koda),
- jednostavnost u odnosu na postojeći objektni model.

Ipak, usporedimo li osnovne koncepte objektnog i skriptnom modela, ustanovit ćemo da su koncepti usporedivi i da među njima postoji određeni stupanj kompatibilnosti.

4.5.1 Klasa - metaskripta

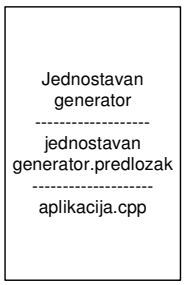
Klasa iz objektnog modela predstavlja tip objekta (=određuje njegove atribute, metode i prava pristupa). Klasa je u dijagramu klasa (engl. Class Diagram)

predstavljena pravokutnikom, koji sadrži naziv klase, te atribute i metode, zajedno s pravima pristupa (slika 4.19).



Slika 4.19:Element klasa u dijagramu klasa

Metaskripta iz skriptnog modela predstavlja metaprogram, predložak koda koji se koristi za generiranje. Metaskripta je u dijagramu metaskripata predstavljena pravokutnikom (slika 4.20), koji sadrži naziv metaskripte, izvor programskog koda i (opcionalno) izlaz programskog koda (npr. naziv izlazne datoteke).



Slika 4.20:Element metaskripta u dijagramu metaskripata

Za razliku od dijagrama klasa, dijagram metaskripata je definiran kao struktura tipa stablo, koja se sastoji od jednorazinskih generatora i višerazinskih generatora koji se dobiju superpozicijom jednorazinskih generatora.

4.5.2 Objekt - skripta

Objekt predstavlja instancu **pojedine** klase, relativno neovisnu logičku cjelinu unutar programa s vlastitim svojstvima (atributima) i funkcionalnostima (metodama).

Skripta (=aplikacija, generirani programski kod) predstavlja proizvod generiranja, odnosno aplikaciju ili dio aplikacije, koji je definiran s **jednom ili više** metaskripata, te vezama i izvorima podataka.

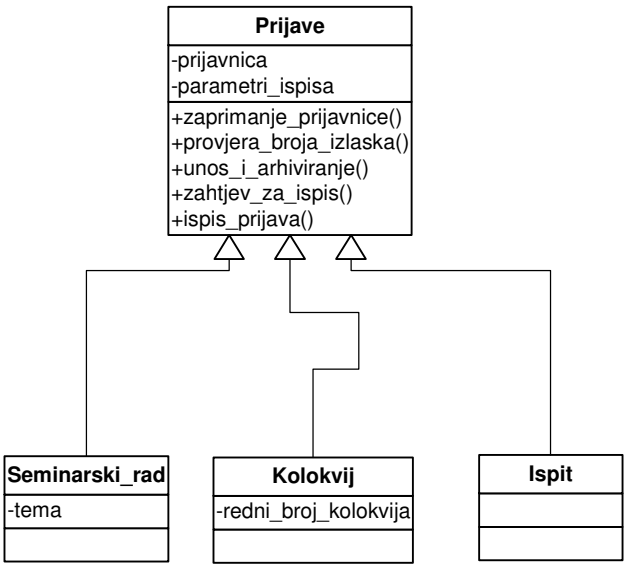
4.5.3 Enkapsulacija

Enkapsulacija u objektnom modelu predstavlja integraciju podataka i funkcionalnosti unutar klasa, odnosno njihovih objekata.

U skriptnom modelu enkapsulacija postoji u opisu aplikacije. Opis aplikacije enkapsulira zadana svojstva (aspekte) aplikacije na razini pojedinih grana dijagrama parametara opisa aplikacije. Svojstva se mogu odnositi i na podatke i na funkcionalnosti.

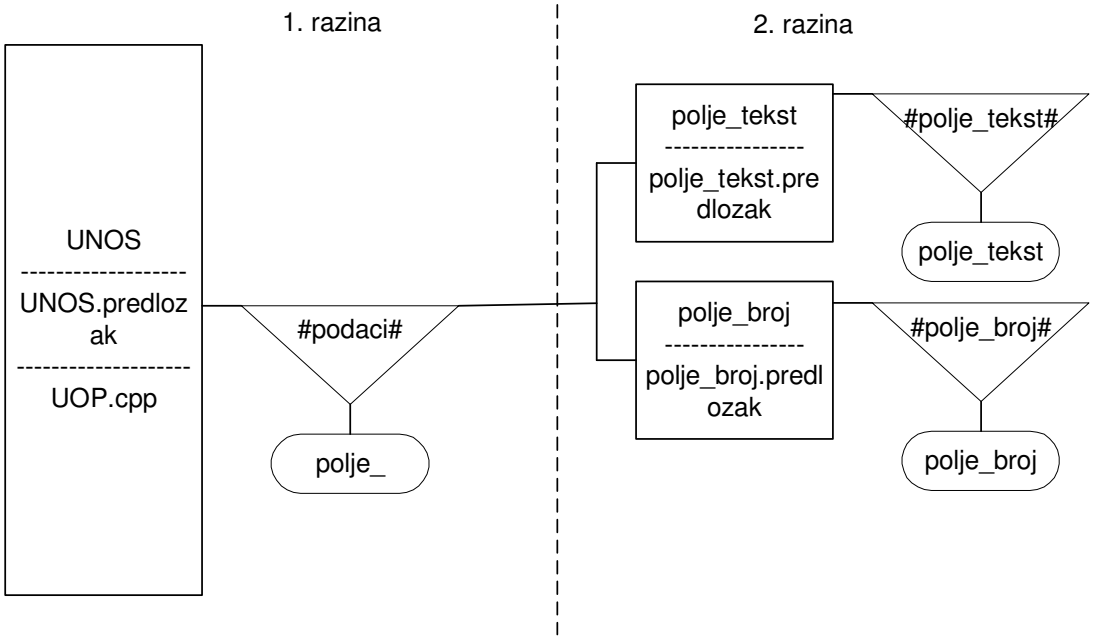
4.5.4 Nasljeđivanje

Nasljeđivanje - predstavlja koncept u okviru objektnog modela prema kojem se dio svojstava i funkcionalnosti nadređene klase prenosi na podređenu klasu (slika 4.21).



Slika 4.21: Primjer nasljeđivanja u dijagramu klasa: klase Seminarski_rad, Kolokvij i Ispit nasljeđuju klasu Prijave

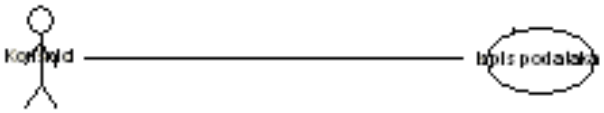
Nasljeđivanje u skriptnom modelu funkcionira tako da se nadređena metaskripta proširuje svojstvima i funkcionalnostima podređenih metaskripata i izvora podataka (slika 4.22), za razliku od objektnog modela, gdje podređene klase nasljeđuju nadređene.



Slika 4.22: Nadređena metaskripta proširuje se svojstvima i funkcionalnostima podređenih metaskripata i izvora podataka (detalj iz dijagrama metaskripata).

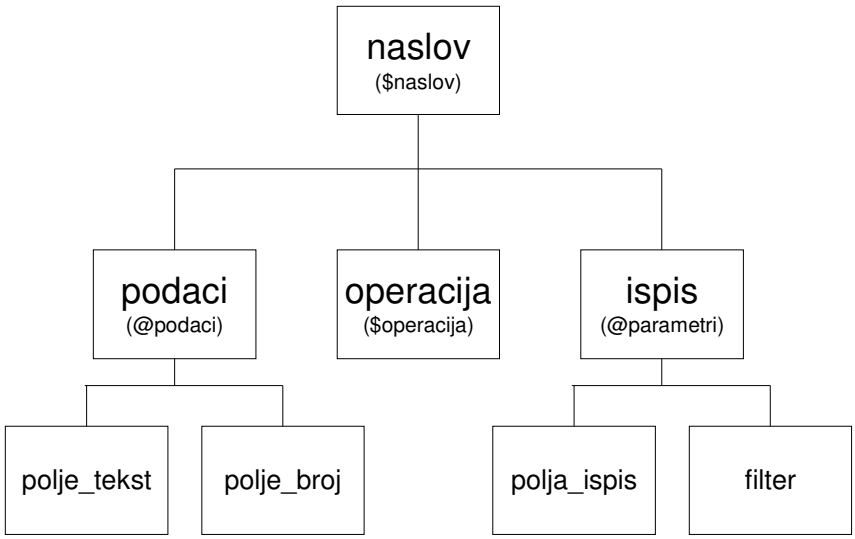
4.5.5 Dijagram slučajeva upotrebe - dijagram parametara opisa aplikacije

Dijagram slučajeva primjene (eng. Use Case Diagram) – definira sve funkcionalnosti aplikacije u odnosu na krajnjeg korisnika (slike 4.23).



Slika 4.23: Pojedini slučaj primjene iz dijagrama slučajeva primjene

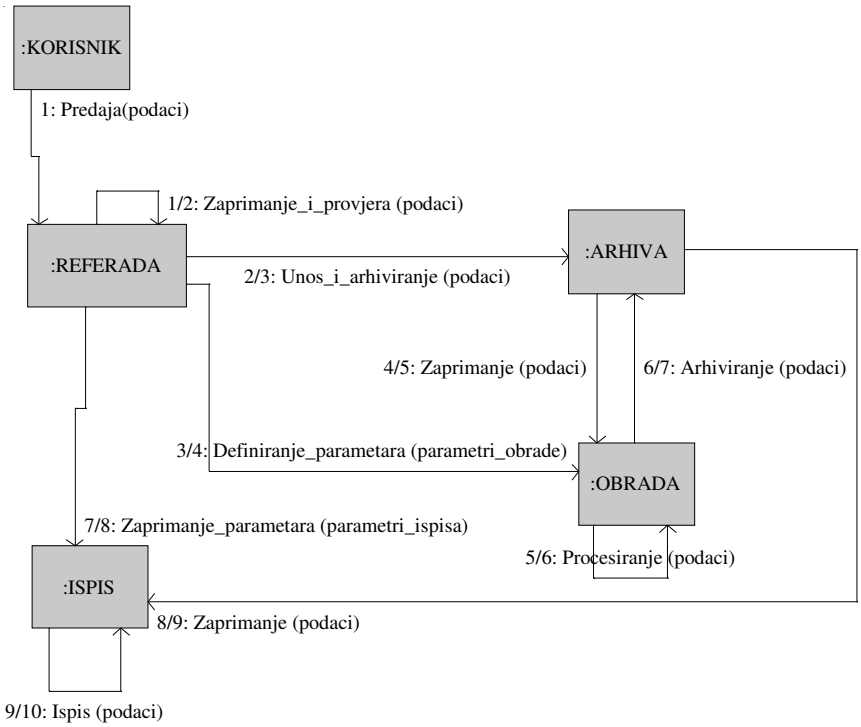
Dijagram parametara opisa aplikacije – definira samo pojedine aspekte aplikacije, one po kojima je ta aplikacija specifična u okviru svoje problemske domene (slika 4.24). Funkcionalnosti su zadane na nižoj razini apstrakcije, unutar metaskripata.



Slika 4.24: Primjer dijagrama parametara opisa aplikacije

4.5.6 Dijagram suradnje - dijagram metaskripata

Dijagram suradnje (eng. Collaboration Diagram) – prikazuje dinamičku strukturu slučaja primjene temeljenu na klasama, interakcijama i porukama (slika 4.25). Pojedini slučaj primjene može se promatrati kroz više razina složenosti, tako da postoji njihova hijerarhijska struktura.



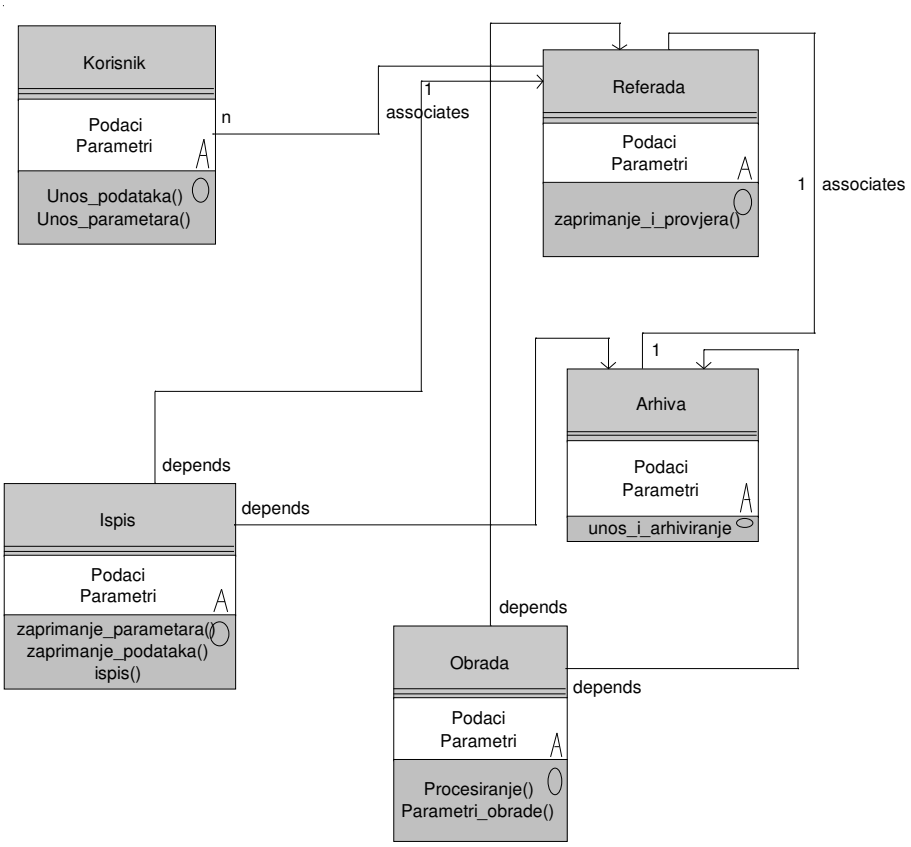
Slika 4.25: Primjer dijagrama suradnje

Dijagramu metaskripata odnosi se na generator - poruke su predstavljene izvorima podataka, klasama odgovaraju metaskripte, dok interakcije predstavlja element veze (slika 4.26).
Dijagram metaskripata također se može promatrati kroz više razina složenosti, tako da svako podstablo predstavlja zasebni dijagram metaskripata.

Slika 4.26: Elementi dijagrama metaskripata

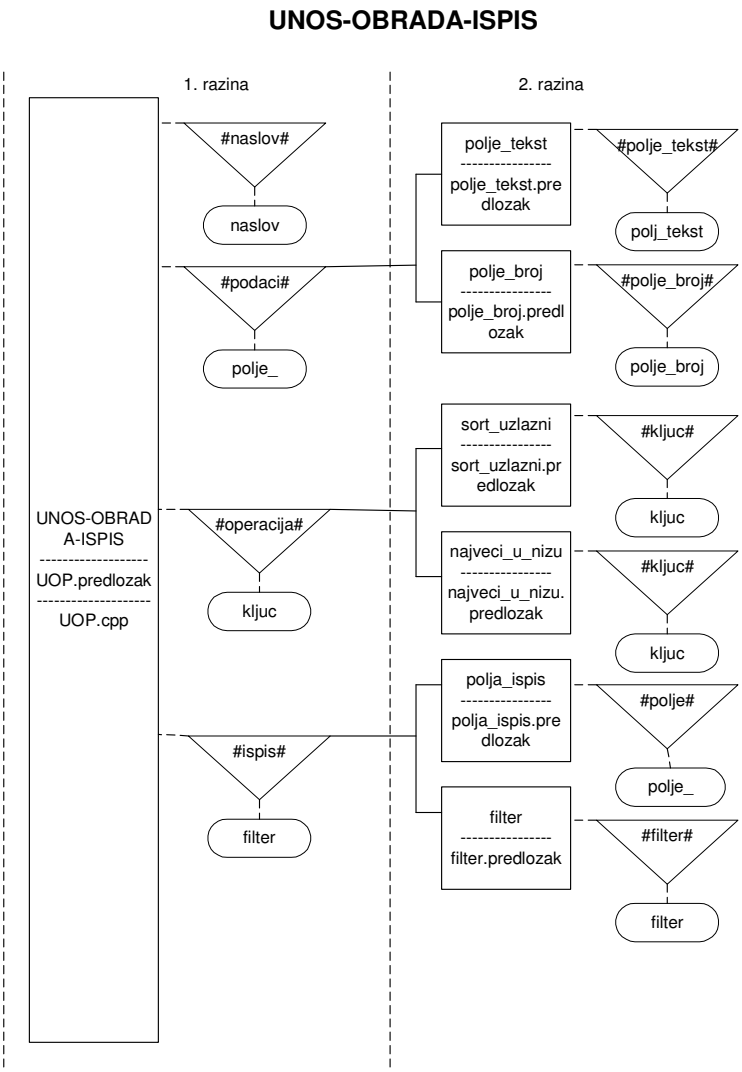
4.5.7 Dijagram klasa - dijagram metaskripata

Dijagram klasa (eng. Class Diagram) – prikazuje skup atributa i operacija definiranih unutar klasa koje su u međusobnoj vezi za pojedinu aplikaciju, odnosno njezine slučajeve primjene (slika 4.27).



Slika 4.27: Primjer dijagrama klasa

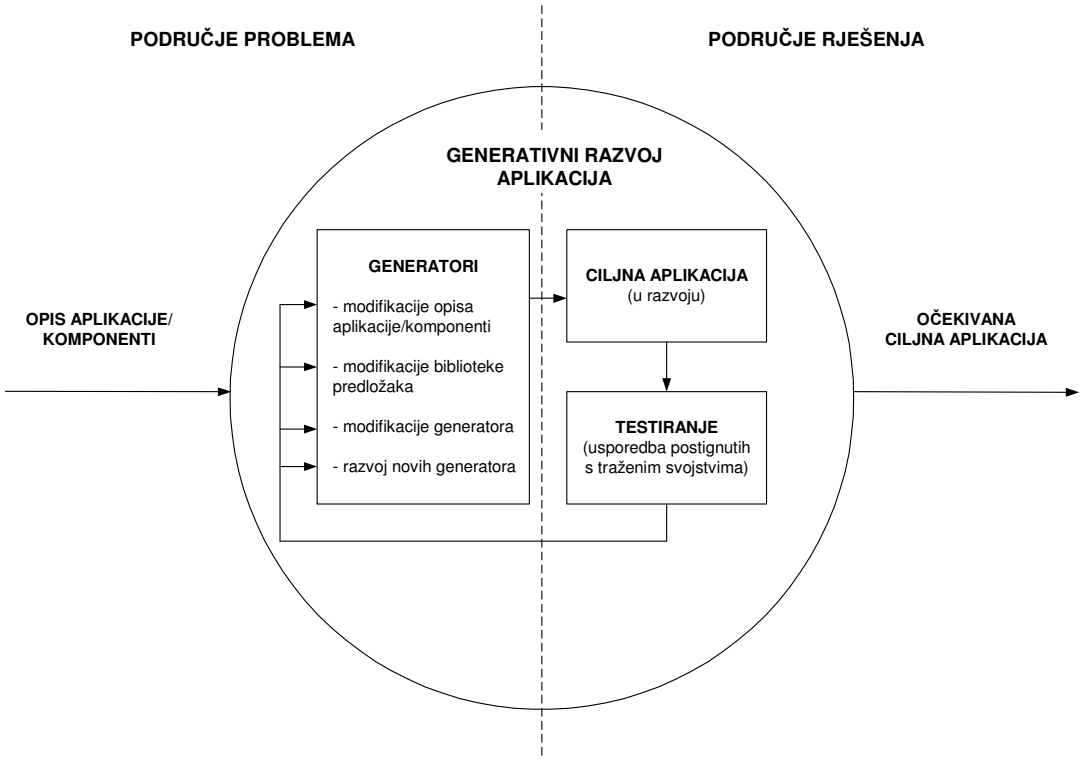
Dijagram metaskripata prikazuje metaskripte i njihove veze s drugim metaskriptama i izvorima podataka (slika 4.28). Može se definirati na razini aplikacije ili pojedine njene komponente.



Slika 4.28: Primjer dijagrama metaskripata

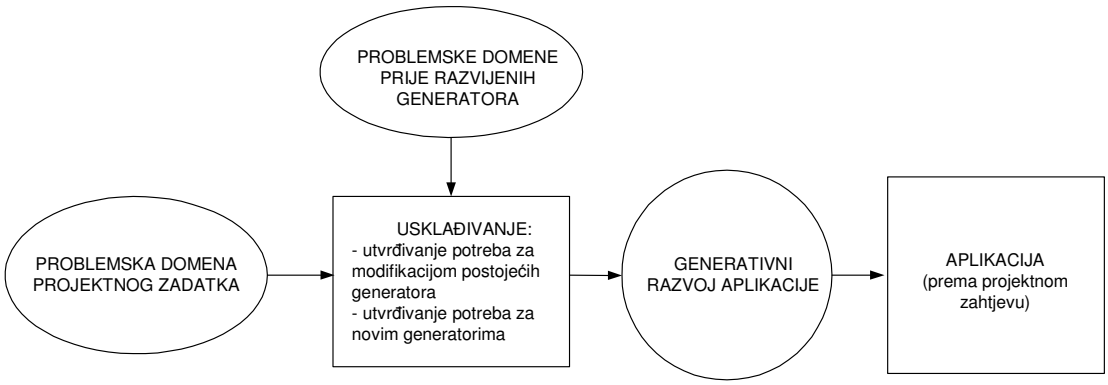
5 GENERATIVNI RAZVOJ APLIKACIJA

Temeljna značajka generativnog programiranja u odnosu na ostale discipline automatskog programiranja je mogućnost korištenja više generatora za izradu aplikacija (Czarnetzki i Eisenecker, 2000.(2), str. 4). To znači da se generatori mogu izrađivati i za generiranje pojedinih komponenti, a ne nužno za kompletne aplikacije. Generativno programiranje prema predloženom konceptu, temeljenom na jezicima skripata omogućuje još veću fleksibilnost u razvoju generatora i aplikacija, kao kružnom procesu (slika 5.1).



Slika 5.1: Razvoj generatora i aplikacija kao kružni proces

Prema tome, generativni razvoj aplikacija uključuje i izradu, te modifikacije generatora, kako bi se oni prilagodili potrebama zadane aplikacije. No, tako razvijeni generatori mogu se kasnije koristiti u okviru novih projekata, odnosno, gotovi generatori predstavljaju ujedno i bazu znanja o odgovarajućim problemskim domenama, koje je potrebno uskladiti s problemskom domenom projektnog zadatka (slika 5.2):



Slika 5.2: Usklađivanje problemske domene prije razvijenih generatora s problemskom domenom projektnog zadatka

Dakle, u prvoj fazi rješavanja novog projektnog zadatka utvrđuje se primjenjivost postojećih generatora, te u skladu s tim, potrebe za modifikacijama postojećih generatora i razvojem novih.

5.1 Pregled do sada razvijenih generatora

Do sada razvijeni generatori aplikacija bit će predstavljeni pomoću odgovarajućih skriptnih modela, te primjerom aplikacije. Ti generatori izrađeni su najprije za razvoj web aplikacija u Perl-u, ali su također poslužili i kao osnova za razvoj novih generatora, što je detaljnije obrađeno u poglavlju 6.2.

5.1.1 Generator web aplikacija za daljinsko održavanje baza podataka

Generator web aplikacija za daljinsko održavanje baza podataka omogućuje, na temelju jednostavnog opisa strukture tablica i veza između tih tablica u bazi podataka, kreiranje skupa CGI skripti u programskom jeziku Perl, te obrazaca u HTML-u, za osnovne operacije održavanja baza podataka pomoću web sučelja.
Funkcionalnosti i način implementacije aplikacija za daljinsko održavanje baza podataka date su u tablici 5.1:

Funkcionalnost	Implementacija
glavna stranica s hipervezama na ostale funkcionalnosti	HTML dokument
kreiranje tablice u bazi podataka	skripta za kreiranje tablice u bazi podataka
ispis sadržaja tablice	skripta za ispis
unos novog sloga u tablicu	obrazac u HTML-u skripta za prikaz obrasca za unos podataka skripta za unos sadržaja obrasca u tablicu
ispravak sloga u tablici	obrazac u HTML-u skripta za prikaz obrasca za ispravak podataka skripta za ispravak sadržaja u tablici
brisanje sloga iz tablice	skripta za ispis sadržaja sloga skripta za brisanje sloga iz tablice

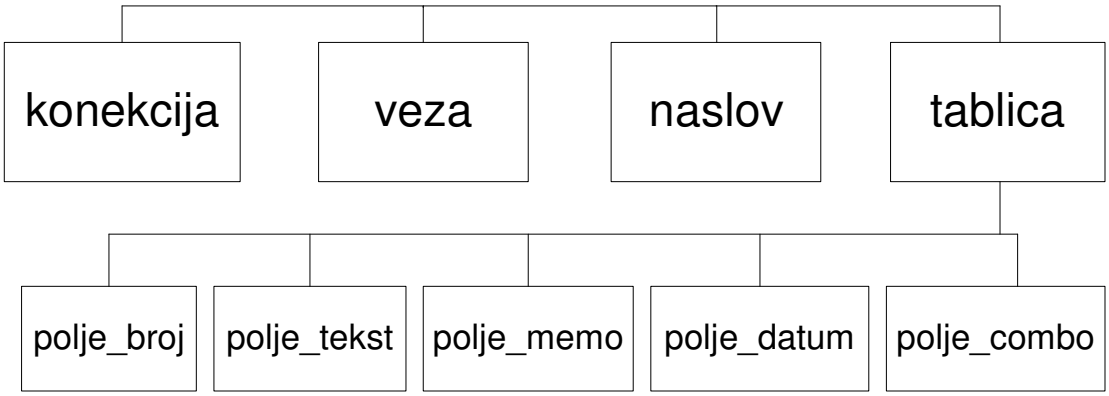
Tablica 5.1: Funkcionalnosti i način implementacije aplikacija za daljinsko održavanje baza podataka

5.1.1.1 Skriptni model generatora aplikacija za daljinsko održavanje baza podataka

Dijagram parametara opisa aplikacija za daljinsko održavanje baza podataka definira sve elemente opisa aplikacija, te njihovu hijerarhiju. U slučaju generatora aplikacija za daljinsko održavanje baza podataka potrebno je definirati slijedeće elemente opisa:

- konekcija na bazu podataka - sadrži instrukciju ciljnog jezika za konekciju na bazu podataka (naziv servera, naziv baze podataka i korisničkog računa na bazi podataka)
- veze između tablica - definiraju se navođenjem obje tablice i ključem za povezivanje
- grupe za održavanje pojedinih tablica unutar baze podataka. Svaka takva grupa sadrži slijedeće podatke:
    - naslov: naziv grupe
    - tablica: naziv tablice u bazi podataka
    - popis polja u tablici baze podataka: polja mogu biti numerička, tekstualna (kratki i dugi znakovni nizovi), datumska, te strani ključevi - veze prema drugim tablicama

Odgovarajući dijagram parametara opisa aplikacija je slijedeći (slika 5.3):



Slika 5.3: Dijagram parametara opisa aplikacija za daljinsko održavanje baza podataka

Primjer: opis aplikacije za održavanje sadržaja tri tablice: predmeti, rokovi i predlosci.

konekcija:$DBH=DBI->connect ("dbi:Pg:dbname=darados;host=alf.cat.foi.hr","darados","");
veza:predmeti->rokovi.sifra_predmeta

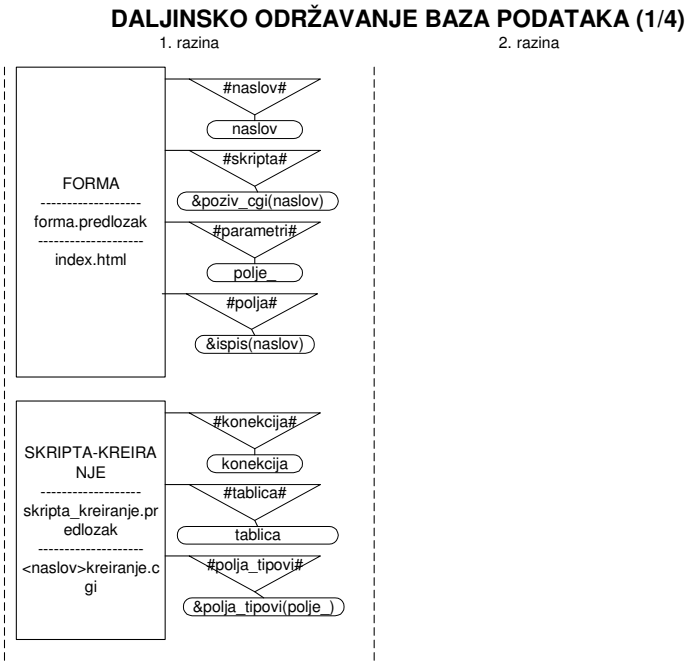
naslov:predmeti
tablica:proba_predmeti2
polje_tekst:naziv_predmeta
polje_broj:godina_studija

naslov:rokovi
tablica:proba_rokovi2
polje_combo:sifra_predmeta,tablica:proba_predmeti2,kljuc:r_broj,polje:naziv_predmeta,vrijed
nost:rokovi
polje_datum:datum_ispita
polje_tekst:vrijeme_ispita
polje_broj:dvorana
polje_memo:rezultati

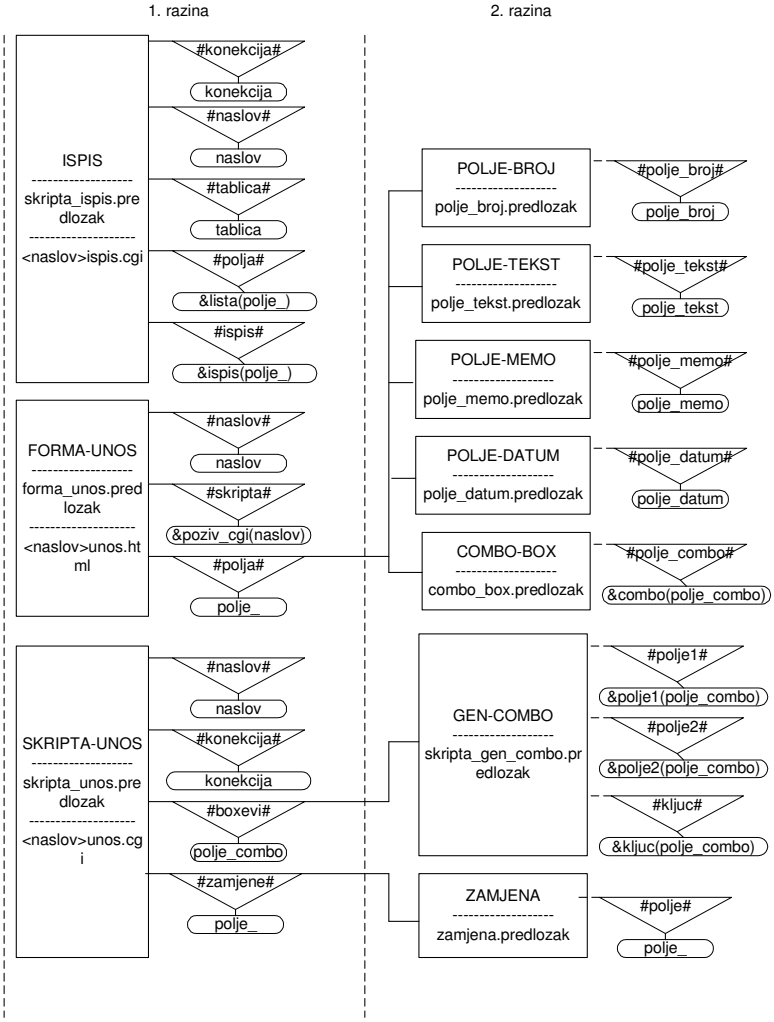
naslov:predlosci
tablica:proba_predlosci
polje_tekst:naziv_predloska
polje_memo:predlozak

U primjeru je definirana konekcija na bazu podataka, veza između tablice predmeta i tablice naslova, te struktura tablica predmeta, naslova i predložaka.

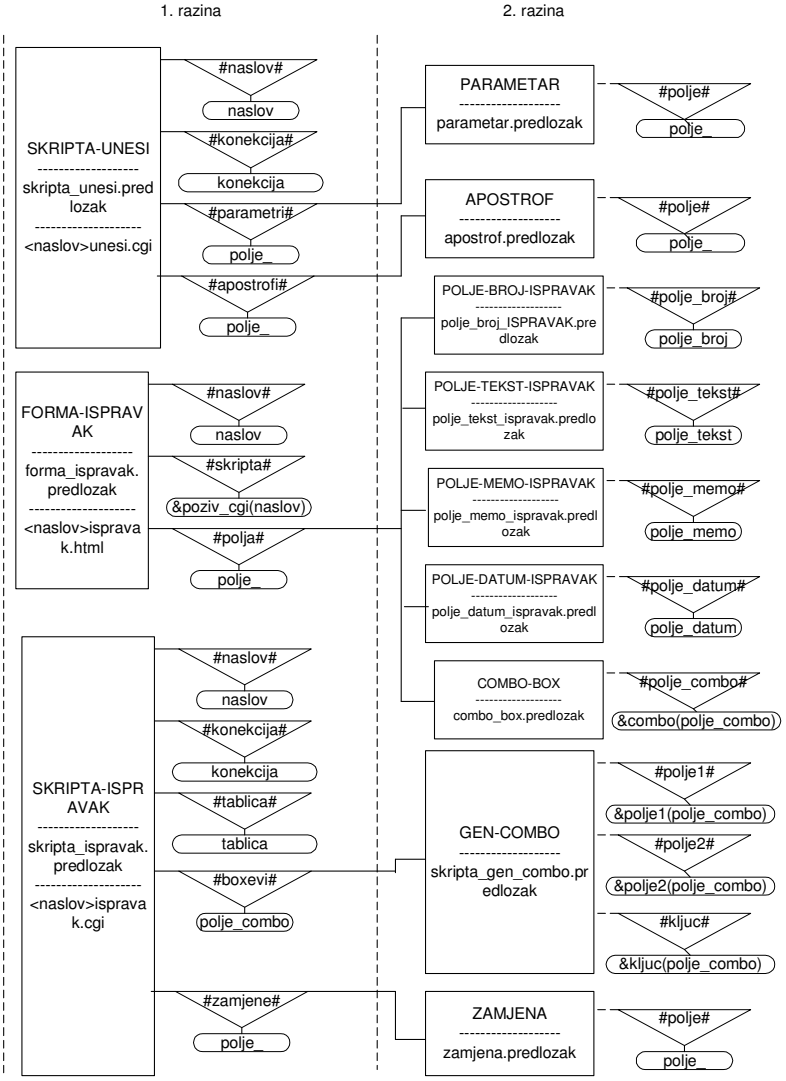
Dijagram metaskripata generatora aplikacija za daljinsko održavanje baza podataka sadrži za svaku implementaciju iz tablice 5.1 po jednu metaskriptu na 1. razini, te se za svaku od njih generira odgovarajući programski kod - CGI skripta ili obrazac u HTML-u (slika 5.4):

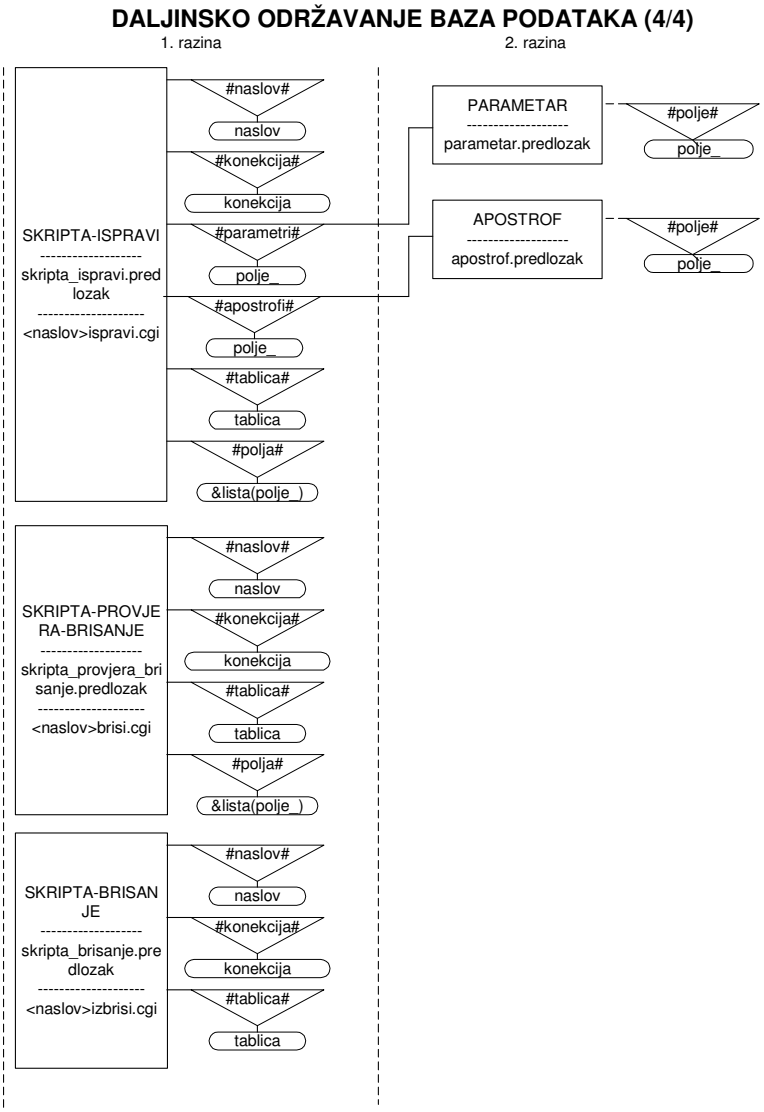


DALJINSKO ODRŽAVANJE BAZA PODATAKA (2/4)



DALJINSKO ODRŽAVANJE BAZA PODATAKA (3/4)





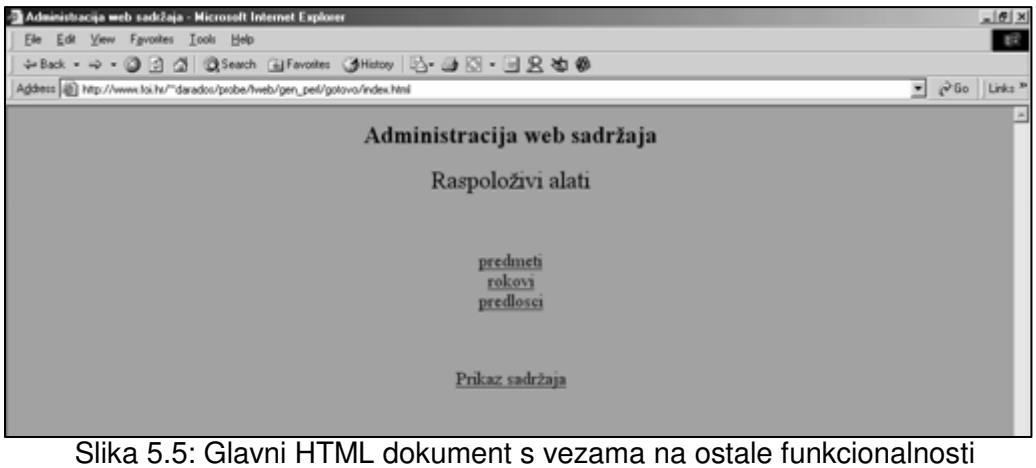
Slika 5.4: Dijagram metaskripata generatora aplikacija za daljinsko održavanje baza podataka

Možemo vidjeti da dijagram metaskripata generatora aplikacija za daljinsko održavanje baza podataka definira 11 generatora, koji, međutim, koriste isti opis aplikacije.

5.1.1.2 Primjer generirane aplikacije za daljinsko održavanje baza podataka

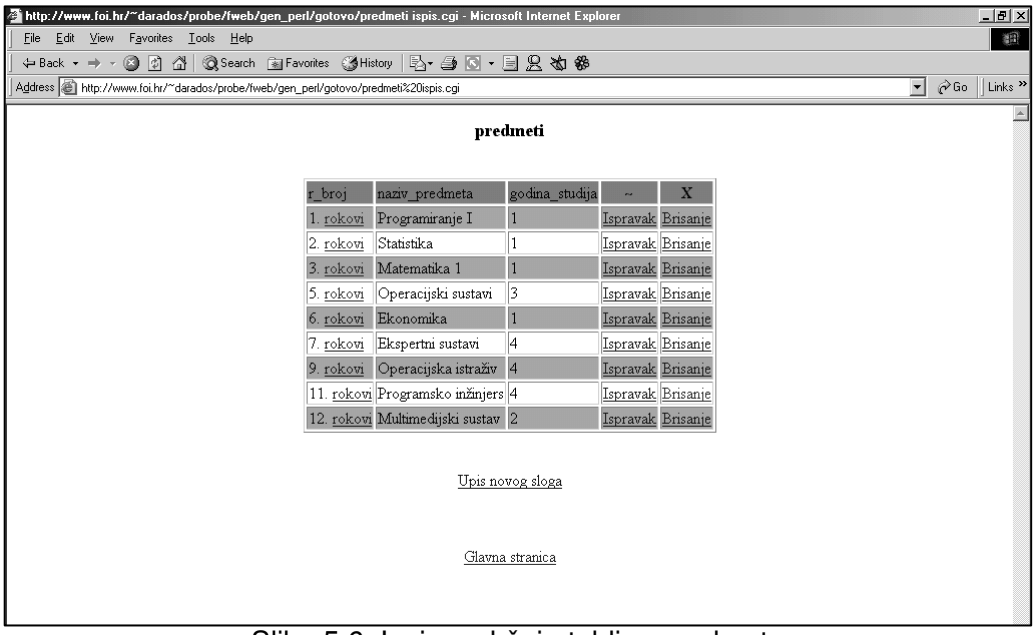
Prema odgovarajućem opisu aplikacije, datom u poglavlju 5.1.1.1. generirana je jednostavna aplikacija za održavanje sadržaja tri tablice u bazi podataka: predmeti, rokovi i predlosci.

Slika 5.5. prikazuje glavni HTML dokument s hipervezama na ostale funkcionalnosti, koje su također generirane iz istog opisa aplikacije, osim mogućnosti 'Prikaz sadržaja' koja se generira pomoću generatora korisničkih web aplikacija.

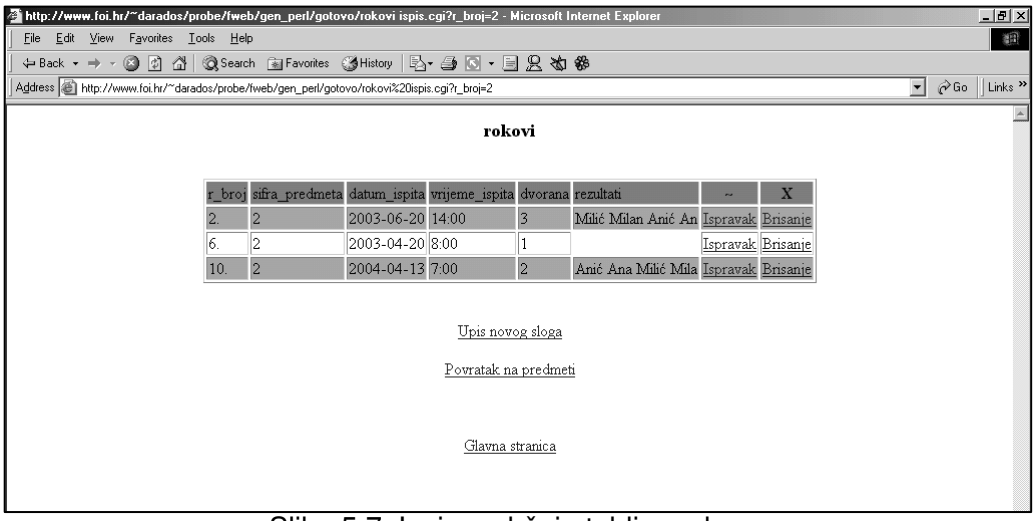


Slika 5.5: Glavni HTML dokument s vezama na ostale funkcionalnosti

Slike 5.6.i 5.7. i 5.8. prikazuju ispis sadržaja tablice predmeta, odnosno tablice rokova.

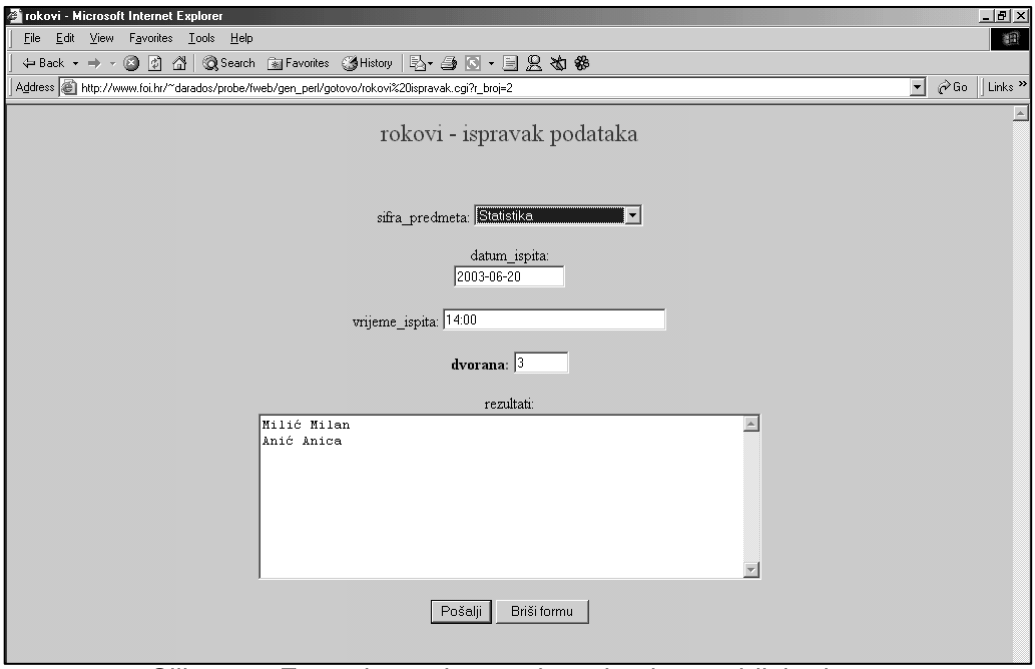


Slika 5.6. Ispis sadržaja tablice predmeta



Slika 5.7. Ispis sadržaja tablice rokova

Slika 5.8 prikazuje formular za ispravak podataka u tablici rokova.



Slika 5.8. Formular za ispravak podataka u tablici rokova

Prema tome, generator aplikacija za daljinsko održavanje baza podataka može se koristiti za održavanje sadržaja potrebnih za funkcioniranje korisničkih web aplikacija, koje se također mogu izraditi pomoću odgovarajućeg generatora, obrađenog u poglavlju 5.1.2.

5.1.2 Generator korisničkih web aplikacija

Generator korisničkih web aplikacija koristi se za izradu aplikacija u obliku CGI skripti u Perlu koje podatke iz baze podataka prezentiraju u obliku web stranica. Za prikaz web stranica koriste se odgovarajući predlošci u HTML-u, koji sadrže odgovarajuće oznake na onim pozicijama gdje se pojavljuju sadržaji iz baze podataka.

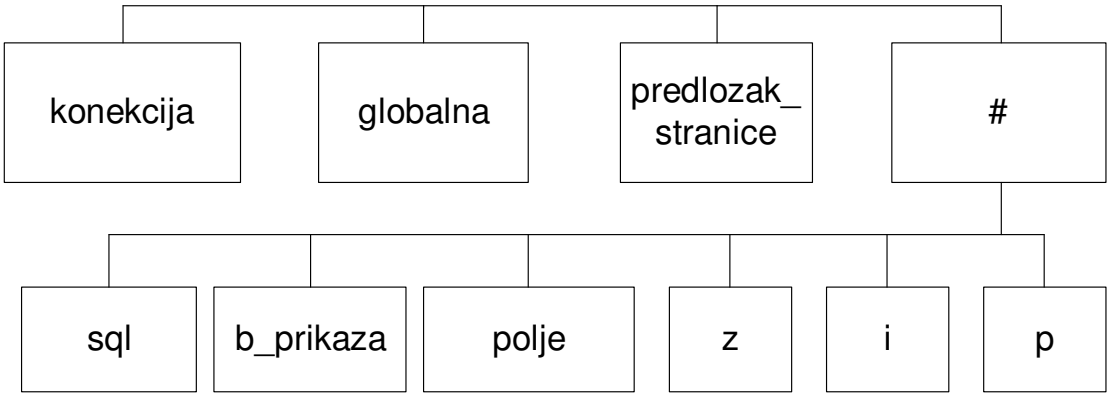
Aplikacija se generira na temelju jednostavnog opisa koji definira parametre pomoću kojih se određuju sadržaji koji će biti prezentirani na web stranici, SQL upiti za učitavanje odgovarajućeg predloška stranice, te podaci za prezentiranje na zadanim pozicijama. Moguće je uključiti i programski kod u Perl-u koji omogućuje obradu podataka prije njihovog prezentiranja na web stranici.
Ovaj generator najčešće se u izradi web aplikacija primjenjuje u kombinaciji s generatorom web sadržaja za daljinsko održavanje baza podataka.

5.1.2.1 Skriptni model generatora korisničkih web aplikacija

Dijagram parametara opisa generatora korisničkih web aplikacija definira elemente i hijerarhiju opisa odgovarajućih aplikacija. Takav opis sadrži slijedeće elemente:

- konekcija na bazu podataka - sadrži instrukciju ciljnog jezika za konekciju na bazu podataka (naziv servera, naziv baze podataka i korisničkog računa na bazi podataka)
- parametri prikaza podataka - definira parametre pomoću kojih se određuju sadržaji koji će biti prezentirani na web stranici.
- SQL upit za učitavanje odgovarajućeg predloška web stranice. Svi predlošci stranica koje koristi aplikacija nalaze se o odgovarajućoj tablici baze podataka. Svaki predložak sadrži oznake - pozicije koje se zamjenjuju odgovarajućim sadržajima iz baze podataka.
- grupe koje se odnose na pojedine oznake, koje se pojavljuju unutar predložaka stranica. Svaka takva grupa sadrži slijedeće podatke:
    - oznaka - niz znakova obostrano omeđen znakovima '#'
    - SQL upit pomoću kojeg se dolazi do podataka iz baze podataka, na temelju kojih će se oznaka zamijeniti odgovarajućim sadržajem
    - odjeljci za obradu podataka - omogućuju umetanje programskog koda u jeziku Perl. Programski kod razvrstava se u tri skupine:
        - z: programski kod koji se izvršava prije izvršavanja SQL upita
        - i: programski kod koji se odnosi na obradu pojedinog pročitanog zapisa iz baze podataka
        - p: programski kod koji se izvršava nakon izvršavanja SQL upita

Odgovarajući dijagram parametara opisa aplikacija je slijedeći (slika 5.9):



Slika 5.9: Dijagram parametara opisa generatora korisničke web aplikacije

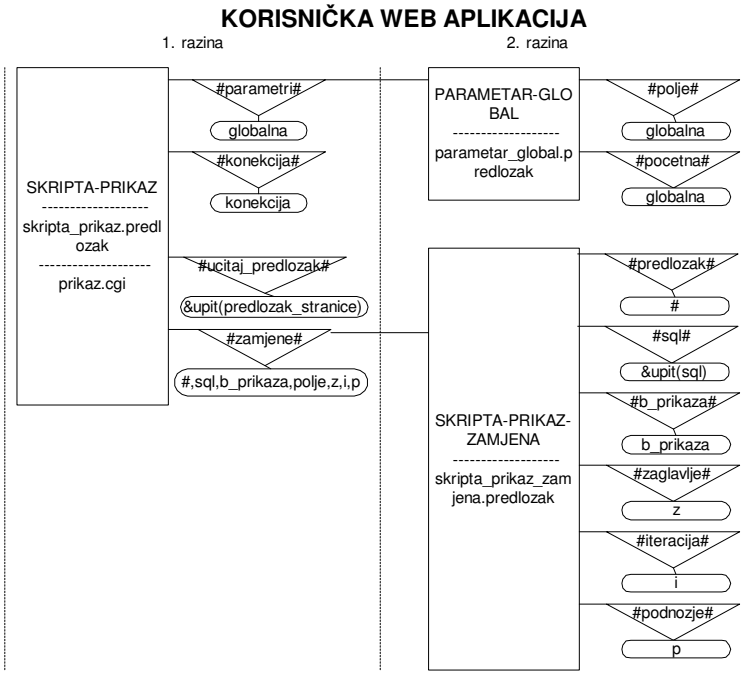
Primjer: opis aplikacije za prikaz ispitnih rokova.

konekcija:$DBH=DBI->connect ("dbi:Pg:dbname=darados;host=alf.cat.foi.hr","darados","");

```
globalna:vrsta,1
globalna:predmet,1
predlozak_stranice:select predlozak from proba_predlosci where r_broj=$vrsta
++++++++++++++++++++++++++++++++++++++++
#predmeti#
sql:select r_broj,naziv_predmeta from proba_predmeti2 order by r_broj
b_prikaza:10000
polje:r_broj
polje:naziv_predmeta
z:$zamjene="$zamjene<center>";
i:$zamjene="$zamjene<a
href=\"prikaz.cgi?vrsta=2&predmet=$r_broj\">$naziv_predmeta</a><br>";
p:$zamjene="$zamjene</center>";
----------------------------------------
#rokovi#
sql:select r_broj,datum_ispita,vrijeme_ispita from proba_rokovi2 where
sifra_predmeta=$predmet order by r_broj
b_prikaza:10000
polje:r_broj
polje_datum:datum
polje:vrijeme_ispita
z:$zamjene="$zamjene<center>";
i:$zamjene="$zamjene$counter. $datum u $vrijeme_ispita<br>";
p:$zamjene="$zamjene</center>";
p:if ($counter==1){
p:$zamjene="$zamjene <center>Nema rokova!</center>";}
----------------------------------------
#naz_roka#
sql:select naziv_predmeta from proba_predmeti2 where r_broj=$predmet
b_prikaza:1
polje:naziv_predmeta
i:$zamjene="$naziv_predmeta";
----------------------------------------
```

U primjeru je definirana konekcija na bazu podataka, dva ulazna parametra (oznaka 'globalna') zajedno s predloženim vrijednostima, SQL upit za učitavanje predloška stranice, te grupe koje se odnose na pojedine oznake u predlošcima.

Dijagram metaskripata generatora korisničkih web aplikacija sadrži samo jednu metaskriptu na 1. razini, za koju se generira odgovarajući programski kod (slika 5.10):

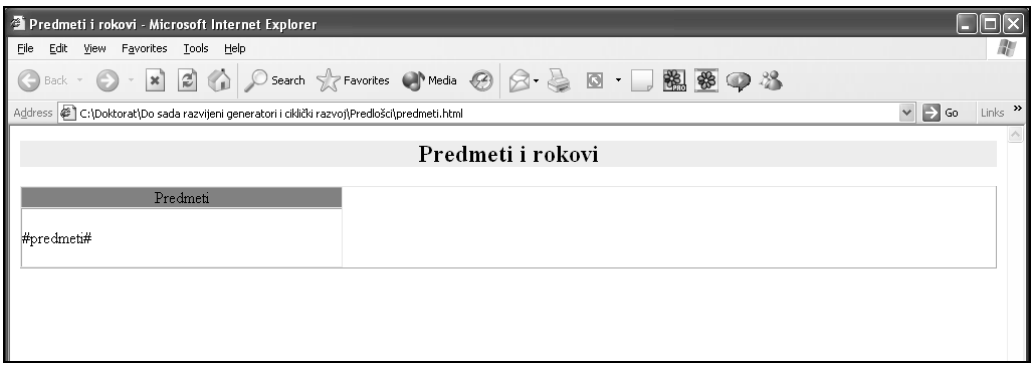


Slika 5.10: Dijagram metaskripata generatora korisničkih web aplikacija

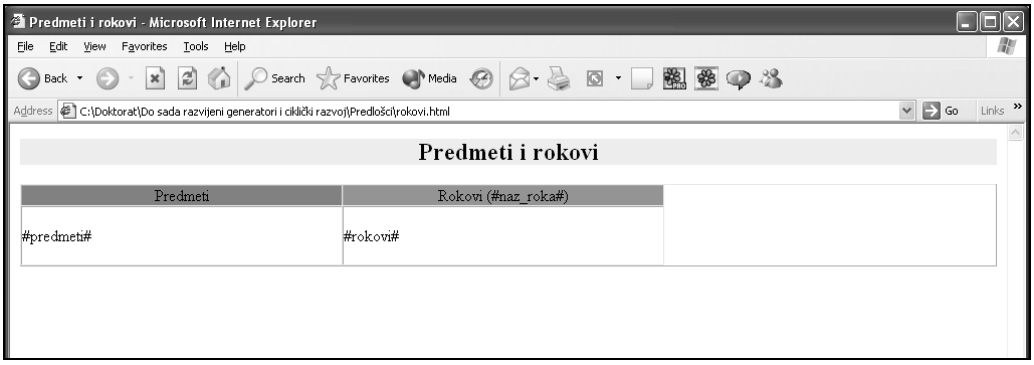
Dijagram metaskripata generatora korisničkih web aplikacija definira samo jedan dvorazinski generator, jer se generira samo jedna programska datoteka.

5.1.2.2 Primjer generirane korisničke web aplikacije

Prema odgovarajućem opisu aplikacije, datom u poglavlju 5.1.2.1. generirana je jednostavna aplikacija za prikaz ispitnih rokova po predmetima. Aplikacija koristi dva HTML dokumenta koji služe kao predlošci za generiranje odgovarajućih web stranica. Predlošci se nalaze unutar tablice 'predlosci' čija struktura je definirana u opisu aplikacije za održavanje baze podataka, datom u poglavlju 5.1.1.1. Slike 5.11 i 5.12 prikazuju predloške stranica:



Slika 5.11:Predložak stranice s popisom predmeta



Slika 5.12:Predložak stranice s popisom ispitnih rokova za zadani predmet

Predložak koji se koristi za prikaz pojedine stranice definiran je oznakom 'predlozak_stranice' u opisu aplikacije. Odgovarajući redak opisa je slijedeći:

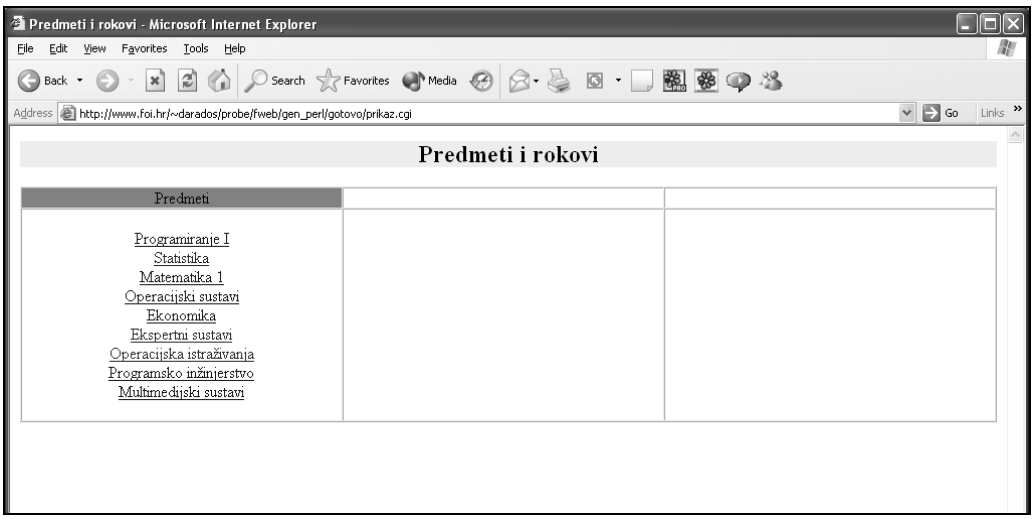
predlozak_stranice:select predlozak from proba_predlosci where r_broj=$vrsta

Dakle, SQL upit učitava sadržaj polja 'predlozak' iz tablice 'proba_predlosci', i to slog s rednim brojem koji je određen vrijednošću varijable '$vrsta'. Varijabla '$vrsta' sadrži vrijednost odgovarajućeg ulaznog parametra, također definiranog unutar opisa aplikacije:

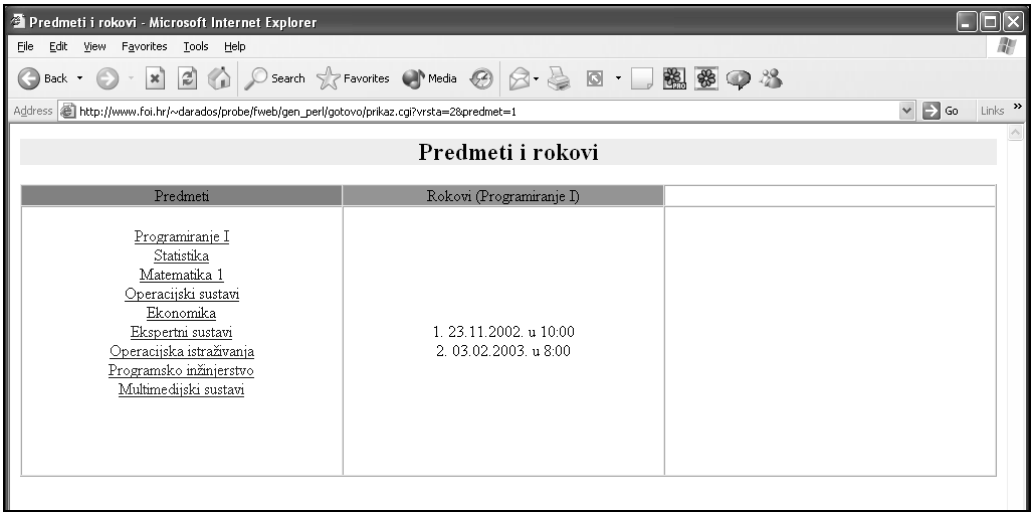
globalna:vrsta,1

Prema tome, ako se ne zada vrijednost parametra 'vrsta', podrazumijevana vrijednost je 1, odnosno, učitava se prvi predložak (predložak stranice s popisom predmeta).

Kod prikaza stranice zamjenske oznake '#predmeti#', '#rokovi#' i '#naz_roka#' zamjenjuju se odgovarajućim sadržajima iz baze podataka, što je također definirano unutar opisa aplikacije. Odgovarajuće primjere prikaza stranica možemo vidjeti na slikama 5.13 i 5.14.



Slika 5.13: Popis predmeta (vrsta=1)



Slika 5.14: Popis ispitnih rokova za zadani predmet (vrsta=2)

Možemo vidjeti da je generator korisničkih web aplikacija usko povezan s generatorom aplikacija za daljinsko održavanje baza podataka, odnosno, cijela aplikacija (administracijski i korisnički dio) izrađuje se korištenjem oba generatora.

5.1.3 Generator anketnih upitnika

Generator anketnih upitnika koristi se za generiranje anketnih upitnika na temelju jednostavnog tekstualnog opisa, koji sadrži pitanja, zajedno s oznakama tipova pitanja i podatke potrebne za kasniji prikaz i obradu rezultata (Dumičić, Sajko i Radošević, 2002., str. 1). Generirane aplikacije (anketni upitnici) sastoje se od CGI skripti u Perl-u i obrazaca u HTML-u.
Funkcionalnosti i način implementacije anketnih upitnika date su u tablici 5.2:

Funkcionalnost	Implementacija
kreiranje tablice u bazi podataka	skripta za kreiranje tablice u bazi podataka
obrazac za unos odgovora	obrazac u HTML-u
unos odgovora u bazu podataka	skripta za unos odgovora u bazu podataka
prikaz rezultata u grafičkom obliku	skripta za prikaz rezultata u grafičkom obliku
prikaz rezultata u tekstualnom obliku	skripta za prikaz rezultata u tekstualnom obliku

Tablica 5.2: Funkcionalnosti i način implementacije anketnih upitnika

5.1.3.1 Skriptni model generatora anketnih upitnika

Dijagram parametara opisa generatora anketnih upitnika definira strukturu odgovarajućeg opisa aplikacije. Opis aplikacije u slučaju generatora anketnih upitnika sadrži slijedeće podatke:

- naziv tablice u bazi podataka koja sadrži podatke iz ispunjenih anketnih upitnika (kod ovog generatora podaci o serveru i bazi podataka nalaze se unutar odgovarajuće metaskripte)
- naslov - naziv ankete koji se pojavljuje na vrhu anketnog upitnika

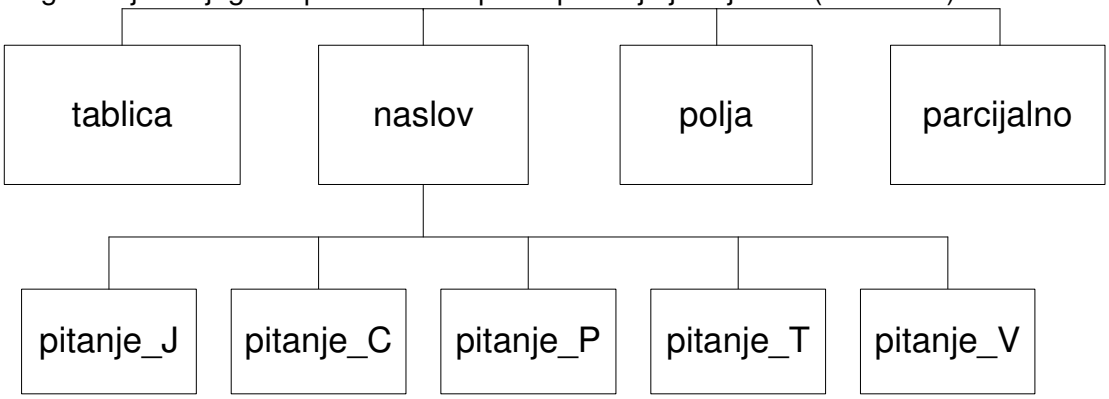
- popis pitanja. Svako pitanje sadrži oznaku tipa, koji definira način prikaza u anketnom obrascu, kao što se vidi u tablici 5.3.

oznaka tipa	prikaz u anketnom obrascu	vrsta kontrole (HTML)
pitanje_J	pitanje s jednim mogućim odgovorom	Radio Button
pitanje_C	pitanje s jednim mogućim odgovorom (prikaz pomoću padajućeg izbornika)	Combo Box
pitanje_P	grafička skala procjene	Scroll Bar
pitanje_T	otvoreno pitanje (tekstualni odgovor)	Text Box
pitanje_V	zatvoreno pitanja s više mogućih odgovora	Check Box

Tablica 5.3: Oznake tipova pitanja i njihov prikaz u anketnom obrascu

- popis polja u tablici baze podataka, koja sadrži odgovore ispitanika
- uvjetne izraze koji definiraju pojedine djelomične rezultate ankete.

Odgovarajući dijagram parametara opisa aplikacija je slijedeći (slika 5.15):



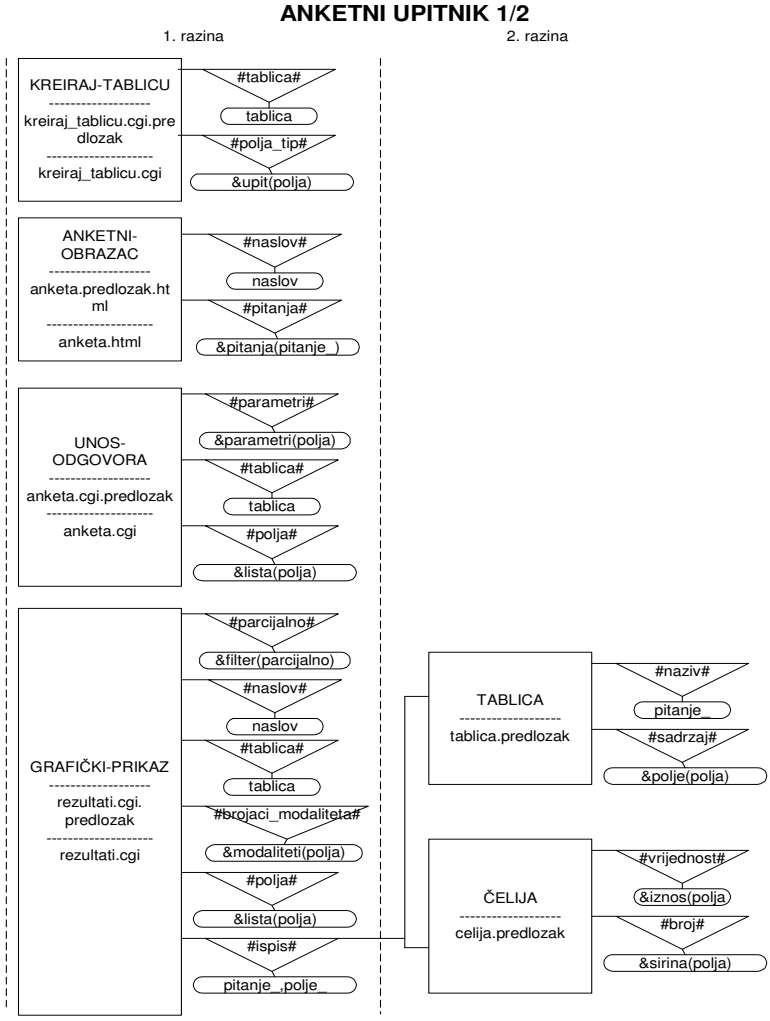
Slika 5.15: Dijagram parametara opisa generatora anketnih upitnika

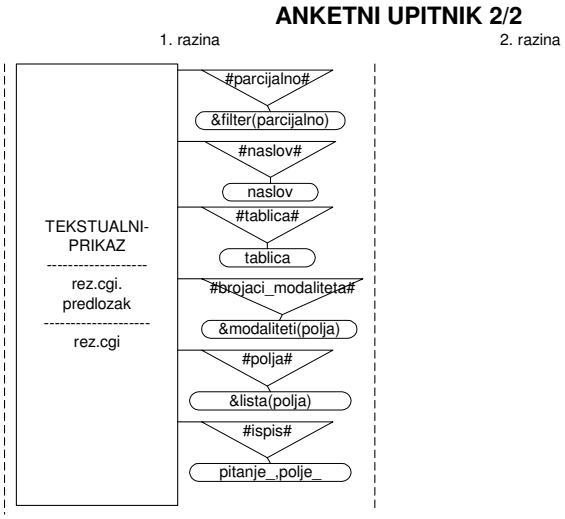
Primjer: opis anketnog upitnika o kućnim ljubimcima.
tablica:anketa_pas
naslov:Moj pas
pitanje_J:Imate li psa?
da
ne
pitanje_C:Koliko je pas star?
štene
do godine dana
1-5 godina
5-10 godina
više od 10 godina
nemam psa
pitanje_V:Što vas sve smeta kod psa?
dlake
smrad
sline

lavež
nešto drugo
pitanje_T:Imate li kakav komentar?
polja:
ima_psa
starost
smeta
napomena1
parcijalno:
ima_psa=1
ima_psa=2
ima_psa=1 and smeta=0
ima_psa=2 and smeta=0

U primjeru je definiran anketni upitnik o psima, koji koristi tablicu 'anketa_pas', ima naslov 'Moj pas', te četiri pitanja različitih tipova. Osim toga, definirana su polja u tablici baze podataka u koja se upisuju odgovori ispitanika, te četiri djelomična rezultata ankete (filtriraju se odgovori ispitanika koji posjeduju, odnosno ne posjeduju psa, te odgovori jednih i drugih koji su odgovorili da ih ništa ne smeta kod psa).

Dijagram metaskripata generatora anketnih upitnika sadrži za svaku implementaciju iz tablice 5.2 po jednu metaskriptu na 1. razini, te se za svaku od njih generira odgovarajući programski kod - CGI skripta ili obrazac u HTML-u (slika 5.16):



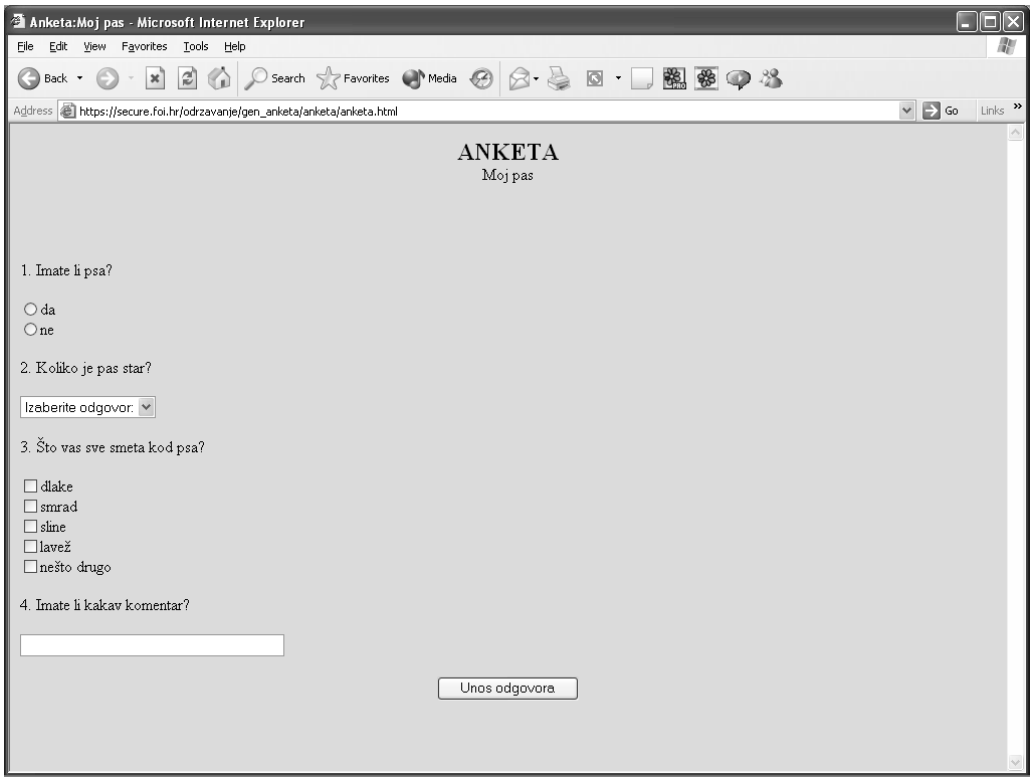


Slika 5.16: Dijagram metaskripata generatora anketnih upitnika

Možemo vidjeti da dijagram metaskripata generatora anketnih upitnika definira pet generatora, koji koriste isti opis aplikacije.

5.1.3.2 Primjer generiranog anketnog upitnika

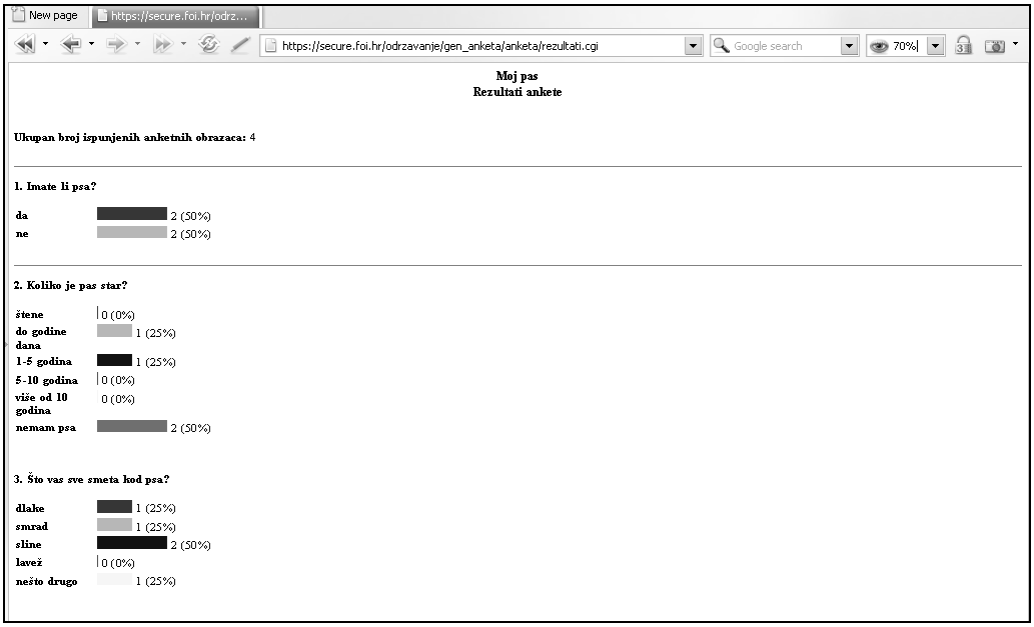
Prema opisu aplikacije iz poglavlja 5.1.3.1. generiran je jednostavan anketni upitnik s četiri pitanja. Slika 5.17 prikazuje odgovarajući anketni upitnik.



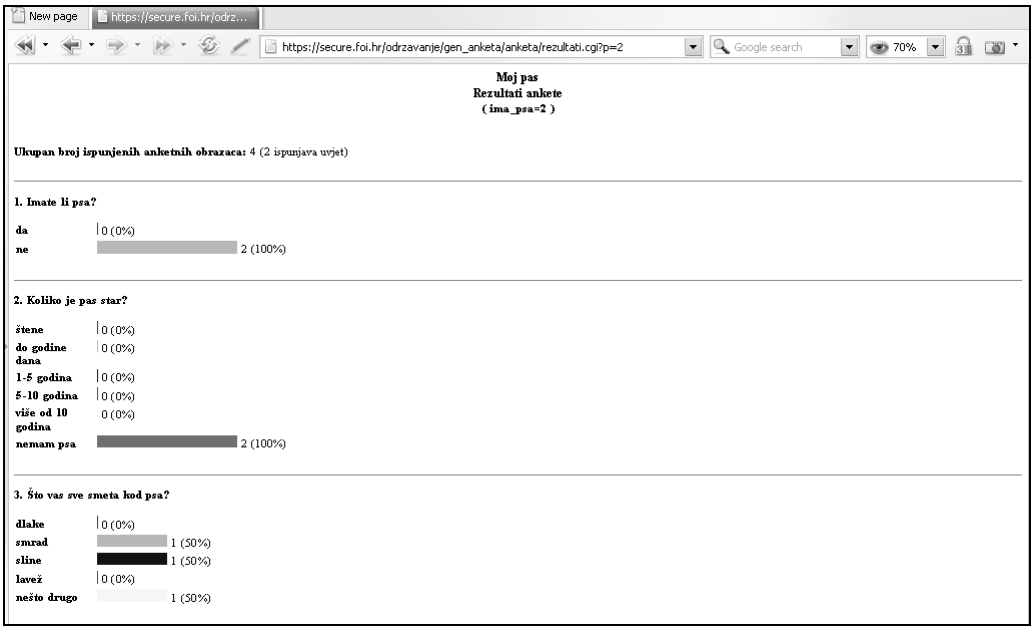
Slika 5.17: Izgled anketnog upitnika

Slika 5.18 prikazuje ukupne rezultate ankete u grafičkom obliku, dok se slika 5.19 odnosi na djelomične rezultate, koji obuhvaćaju ispitanike koji su odgovorili da ne

posjeduju psa (unutar rubrike 'parcijalno' opisa aplikacije ova mogućnost označena je s ima_psa=2).

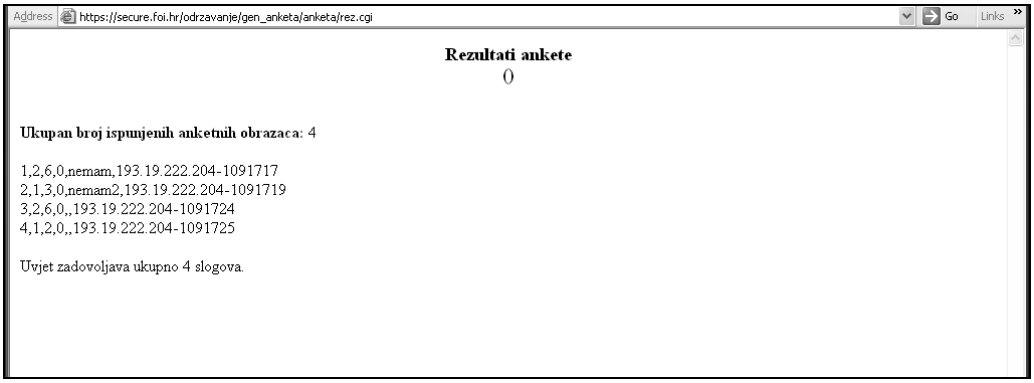


Slika 5.18: Ukupni rezultati ankete u grafičkom obliku



Slika 5.19: Primjer djelomičnih rezultata ankete u grafičkom obliku

Slika 5.20 prikazuje rezultate ankete u tekstualnom obliku, takvom da se može učitati u tablični kalkulator, kao što je Excel.

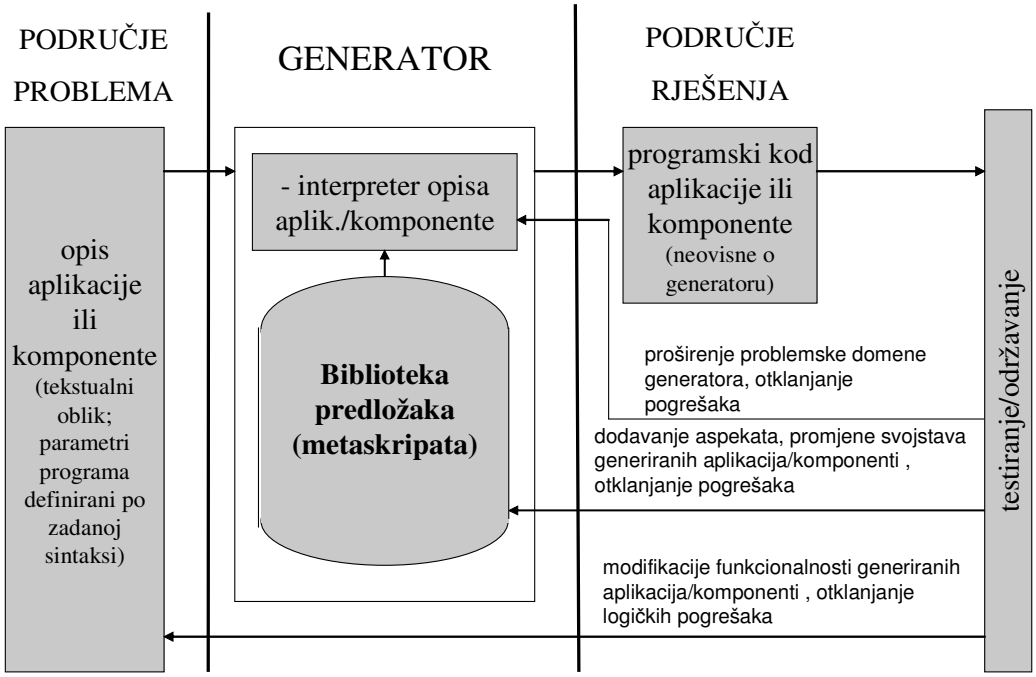


Slika 5.20: Ukupni rezultati ankete u tekstualnom obliku

Prema tome, generator anketnih upitnika može se iskoristiti u okviru raznih istraživanja putem Interneta. Osim mogućnosti ispunjavanja anketnih obrazaca moguća je i jednostavna obrada, te prikaz rezultata.

5.2 Modifikacije aplikacija i generatora

Korištenjem generatora iz poglavlja 5.1 razvijene su različite web aplikacije, za koje su prethodno izrađeni odgovarajući opisi. Daljnji razvoj aplikacija i generatora moguć je kroz modifikacije, koje su moguće na različitim razinama, kao što se vidi na slici 5.21. Opisi prije izrađenih aplikacija služe ujedno kao repozitorij s ugrađenim rješenjima koja se, kao što možemo vidjeti u poglavlju 5.2.1, mogu upotrijebiti u razvoju novih aplikacija ili za prijenos gotovih aplikacija u druga radna okruženja (npr. drugi server ili druga baza podataka).
U nekim slučajevima potrebno je obogatiti generirane aplikacije novim funkcionalnostima, odnosno promijeniti određene detalje implementacije. To se može postići promjenama u predlošcima programskog koda (metaskriptama), što je obrađeno u poglavlju 5.2.2.
Ponekad je potrebno prilagoditi već razvijene generatore, tako da im se modificira problemska domena. Takve prilagodbe su obrađene u poglavlju 5.2.3.



Slika 5.21: Modifikacije aplikacija i generatora na različitim razinama

5.2.1 Modifikacije opisa aplikacije

Modifikacijama opisa aplikacije mijenja se ciljna aplikacija u okviru zadane problemske domene, za koju je izrađen odgovarajući generator. Sintaksa opisa aplikacije zadana je odgovarajućim dijagramom parametara opisa i sastoji se od oznaka i njihovih vrijednosti. Obzirom na jednostavnost opisa, za promjene u nekim slučajevima, kao što je npr. generator anketnih upitnika (obrađen u poglavlju 5.1.3), nisu ni potrebna programerska znanja. Nešto je složeniji opis aplikacije generatora korisničkih web aplikacija (obrađen je u poglavlju 5.1.2).

Složenost opisa aplikacije ovisi o broju mogućih oznaka, za koje se u opisu aplikacije zadaju odgovarajuće vrijednosti. Pojedina svojstva mogu se zadavati u opisu aplikacije, ali i u biblioteci metaskripata. Primjer takvih svojstava je specifikacija baze podataka, koja se u slučaju generatora aplikacija za daljinsko održavanje baza podataka i generatora korisničkih web aplikacija u potpunosti nalazi u opisu aplikacije, dok se u slučaju generatora anketnih upitnika u opisu aplikacije nalazi samo naziv korištene tablice u bazi podataka, dok se ostali podaci potrebni za konekciju na bazu podataka nalaze u odgovarajućoj metaskripti.
Prema tome, veće mogućnosti specifikacije aplikacija proširuju problemsku domenu generatora, tako da postaje moguće generirati aplikacije koje se međusobno više razlikuju ili im je šira mogućnost primjene (npr. u smislu strojnog i programskog okruženja aplikacije). No, s druge strane, takve veće mogućnosti specifikacije dovode do toga da opis aplikacije postaje složeniji.

Osim složenosti, drugo bitno svojstvo opisa aplikacije je **mogućnost uključivanja programskog koda u ciljnom programskom jeziku** u opis aplikacije. Takva mogućnost postoji u opisu aplikacije kod generatora korisničkih web aplikacija i omogućuje precizniju specifikaciju aplikacije, bez potrebe da se u opisu aplikacije definiraju oznake koje se odnose na operacije niže razine (npr. formiranje programskih petlji, aritmetičke operacije i sl.). S druge strane, uključivanjem koda u ciljnom programskom jeziku smanjuje se prenosivost opisa aplikacije, jer se isti više

ne može koristiti za generiranje aplikacije u nekom drugom programskom jeziku. Takva prenosivost postoji u slučaju generatora anketnih upitnika i kod generatora aplikacija za daljinsko održavanje baza podataka.

5.2.2 Modifikacije metaskripata

Programski kod generiranih aplikacija sadržan je najvećim dijelom u metaskriptama, a manjim u opisu aplikacije (ako je dozvoljena mogućnost uključivanja koda u ciljnom programskom jeziku), te eventualno neki detalji programskog koda u funkcijama za predobradu podataka iz opisa aplikacije. Promjene u kodu metaskripata ne utječu na skriptni model aplikacije, ali utječu na njene funkcionalnosti i način implementacije tih funkcionalnosti, uključujući i programski jezik generiranih aplikacija.

Modifikacijama metaskripata moguće je utjecati na svojstva generatora, kao što su:

- uključivanje novih funkcionalnosti i svojstava
- promjena implementacije postojećih funkcionalnosti i svojstava

Prema tome, skriptni model generatora ostaje isti, ali se način implementacije mijenja, čime se postižu nova svojstva generiranih aplikacija, kao što su slijedeća:

- prilagodba na novo strojno i programsko okruženje. Primjeri za to su slijedeći:
    - promjena baze podataka koju aplikacija koristi
    - provjera identiteta korisnika (npr. provjera dvostrukog ispunjavanja kod anketnih upitnika može se riješiti provjerom IP adrese ili pomoću cookia),
- uvođenje novih aspekata aplikacija, npr. dodatne obrade i elementi sigurnosti i
- promjena grafičkog izgleda aplikacije.

Neke promjene većim dijelom zadiru u metaskripte, ali zahtijevaju i manje promjene u kodu generatora, što se u prvom redu odnosi na predobradu podataka iz opisa aplikacije. Primjer za to je promjena programskog jezika generirane aplikacije.

5.2.3 Modifikacije generatora

Modifikacijama postojećih generatora dolazi se do novih generatora, odnosno, do takvih generatora koji generiraju aplikacije iz djelomično promijenjene problemske domene. Naime, osim što se za generiranje složenih aplikacija može koristiti više generatora, moguće je na jednostavan način modificirati i generatore.

Modifikacije generatora moguće su na dva načina:

- kroz modifikacije funkcija za predobradu podataka iz opisa aplikacije i
- kroz modifikacije skriptnog modela generatora.

Modifikacijama funkcija za predobradu podataka iz opisa aplikacije moguće je, zajedno s odgovarajućim promjena u metaskriptama, prilagoditi generator za generiranje aplikacija koje će zadržati funkcionalnosti prethodnih, ali će se njihova implementacija promijeniti. Tipičan primjer je promjena programskog jezika aplikacije. Primjer su generatori web aplikacija za daljinsko održavanje baza podataka, koji su najprije realizirani za generiranje aplikacija u Perl-u, da bi kasnije bili modificirani za generiranje odgovarajućih aplikacija u ASP-u, zadržavajući pritom sve

funkcionalnosti. Modifikacije funkcija za predobradu podataka iz opisa aplikacije bile su potrebne zbog usklađivanja razlika u sintaksi Perl-a i ASP-a, ali je skriptni model takvih generatora ostao je nepromijenjen u odnosu na skriptne modele obrađene u poglavljima 5.1.1.1. i 5.1.2.1.

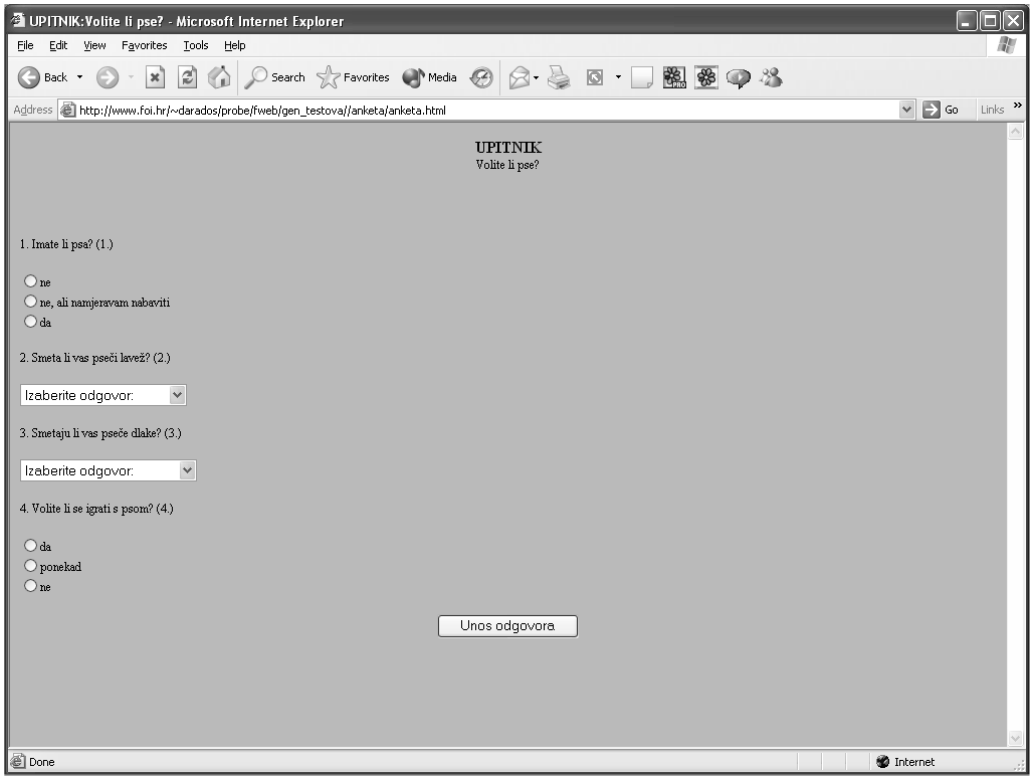
Modifikacije skriptnog modela generatora proizlaze iz potrebe da se opis aplikacija obogati mogućnošću zadavanja novih svojstava, te iz potrebe za promjenom funkcionalnosti generiranih aplikacija koje proizlaze iz tih novih svojstava.

5.2.3.1 Generator web upitnika za testiranje

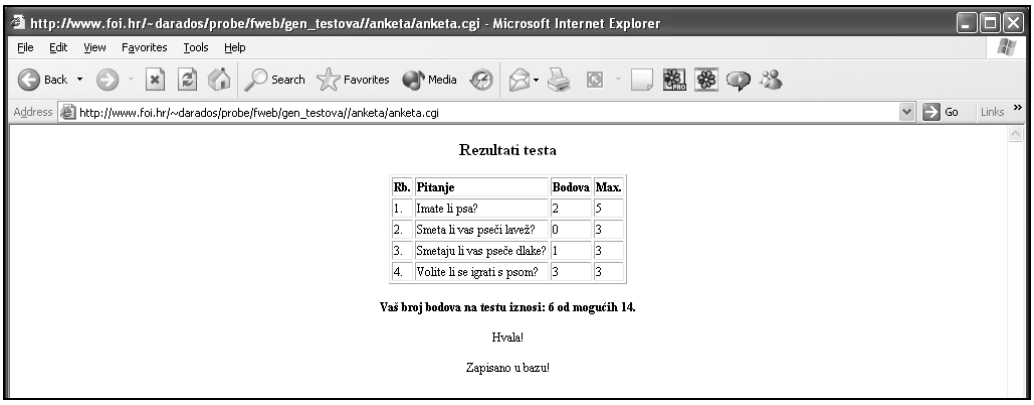
Generator web upitnika za testiranje nastao je modifikacijom generatora anketnih upitnika, dodajući upitnicima mogućnost bodovanja rezultata. Zbog toga je opis aplikacije trebalo proširiti odjeljkom u kojem se definira bodovanje odgovora. U slijedećem primjeru možemo vidjeti opis jednog web upitnika za testiranje:

tablica:upitnik_pas
naslov:Volite li pse?
pitanje_J:Imate li psa?
ne
ne, ali namjeravam nabaviti
da
pitanje_C:Smeta li vas pseći lavež?
ne, uopće me ne smeta
uglavnom me ne smeta
ponekad mi smeta
uglavnom me smeta
jako me smeta
pitanje_C:Smetaju li vas pseće dlake?
ne, uopće me ne smetaju
uglavnom me ne smetaju
ponekad mi smetaju
uglavnom me smetaju
jako me smetaju
pitanje_J:Volite li se igrati s psom?
da
ponekad
ne
polja:
imati
lavez
dlake
igra
bodovi:
imati,5,2,0
lavez,3,1,0,-1,-3
dlake,3,1,0,-1,-3
igra,3,1,-1
parcijalno:

Dakle, opis aplikacije uključuje odjeljak 'bodovanje', koji definira broj bodova za svaki mogući odgovor na pojedino pitanje, kojeg nije bilo kod generatora anketnih upitnika. Odgovarajući obrazac za unos odgovora nije bitno promijenjen u odnosu na generator anketnih upitnika, kao što možemo vidjeti na slici 5.22, ali je uključen prikaz rezultata testa (slika 5.23).



Slika 5.22: Primjer web upitnika za testiranje



Slika 5.23: Primjer rezultata testiranja

Dakle, generator web upitnika za testiranje proširio je funkcionalnost prijašnjeg generatora web upitnika na područje testiranja, čime je otvoreno i novo područje primjene.

6 PROJEKT PRIMJENE GENERATIVNOG RAZVOJA APLIKACIJA

Generativni razvoj aplikacija prema predloženom konceptu generativnog programiranja temelji se na upotrebi više generatora, uz njihove modifikacije na različitim razinama, kao što se može vidjeti na slici 5.1. U ovom poglavlju obrađen je generativni razvoj sustava za anketiranje gostiju hotela, radi dobivanja povratnih informacija potrebnih za unapređenja turističke ponude.

6.1 Projektni zadatak: sustav za anketiranje gostiju hotela

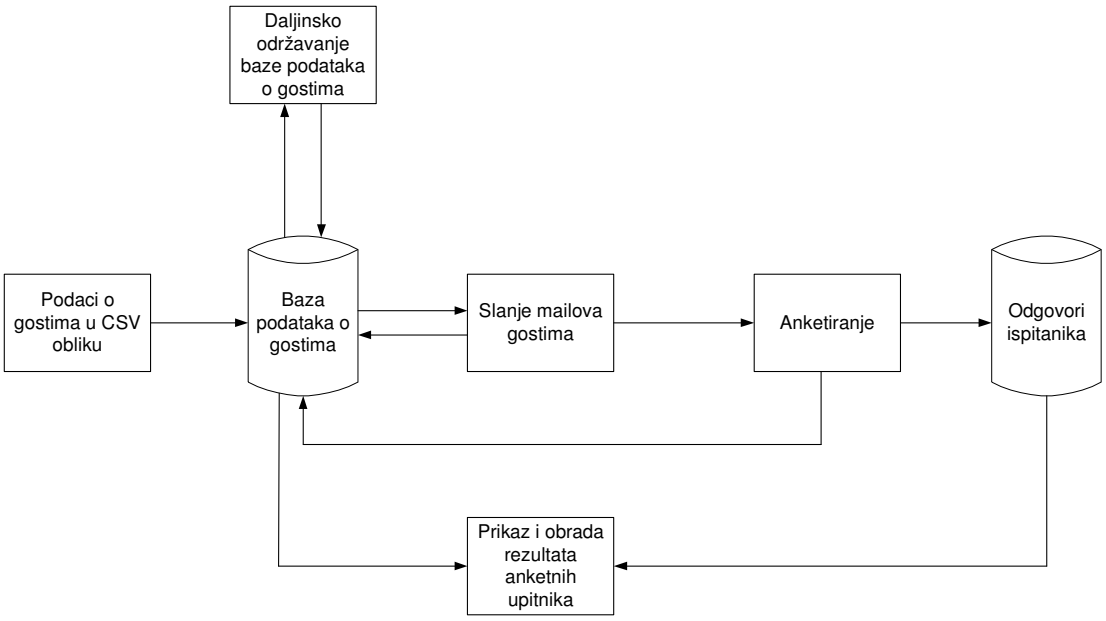
Sustav za anketiranje gostiju hotela treba omogućiti daljinsko održavanje baze podataka preko Interneta o gostima zadanog hotela. Podaci o gostima dostupni su u obliku tablice u Excelu, iz kojeg se mogu konvertirati u CSV format, u kojem se učitavaju u sustav za anketiranje. Svakom gostu, koji posjeduje svoju e-mail adresu, treba poslati e-mail poruku s kratkim tekstom na njemu poznatom jeziku (potrebno je podržati pet jezika: hrvatski, engleski, njemački, talijanski i slovenski) i hipervezom na anketni web upitnik (također na njemu poznatom jeziku). U slučaju da ne ispuni upitnik, potrebno mu je još jednom poslati poruku, ali ukupno ne više dva puta. Rezultati upitnika trebaju biti dostupni u grafičkom i tekstualnom obliku, te trebaju biti uključene zadane obrade. Tekstualni oblik rezultata treba biti pogodan za daljnju obradu u tabličnom kalkulatoru, kao što je Excel.

6.1.1 Zahtijevane funkcionalnosti sustava

Zahtijevane funkcionalnosti sustava za anketiranje gostiju hotela su slijedeće:

- održavanje baze podataka o gostima,
- slanje e-mail poruka gostima s pozivom na ispunjavanje ankete,
- anketiranje gostiju i
- prikaz i obrada rezultata ankete

Odnosno, proces anketiranja gostiju hotela možemo prikazati slijedećim dijagramom (slika 6.1):



Slika 6.1. Proces anketiranja gostiju hotela

U procesu anketiranja (slika 6.1) ostvarene su i potrebne povratne podatkovne veze:

- svaka poslana e-mail poruka mora se evidentirati u bazi podataka o gostima, jer se svakom gostu šalje najviše dva puta
- svaki ispunjeni anketni obrazac mora se evidentirati u bazi podataka o gostima, kako se istima ne bi više slale obavijesti putem e-maila.

6.1.1.1 Održavanje baze podataka o gostima

Održavanje baze podataka o gostima potrebno je realizirati na tri načina:

- učitavanjem podataka iz hotelske evidencije gostiju (u CSV obliku),
- sustavom za daljinsko održavanje baze podataka o gostima (omogućuje ispis podataka o gostima, pojedinačni unos podataka o gostima, te ispravak i brisanje podataka o gostima i
- povratnim vezama od ostalih podsustava (evidencija poslanih e-mail poruka i ispunjenih anketnih upitnika).

Hotelska evidencija gostiju dobiva se od hotela u CSV obliku (pogodnom za obradu u tabličnom kalkulatoru, kao što je Excel). Evidencija sadrži ukupno 82 polja, iz kojih se učitavaju potrebni podaci za anketiranje gostiju, te smještaju u bazu podataka o gostima:

redni_broj_gosta
prezime
ime
spol
e_mail
adresa
telefon
fax
nacionalnost

jezik_za_kontakt
segment_gostiju
datum_rodjenja
dolazak (datum dolaska)
odlazak (datum odlaska)

Osim ovih polja baza podataka gostiju treba sadržavati još i slijedeće podatke:

kontaktiran (šifra kontakta: 1-kontaktiran 1. put, 2-kontaktiran 2. put, 5-odgovorio, 6-ne odgovara, 7-ne želi odgovoriti)
redni_broj_tjedna (izračunava se iz datuma odlaska)
sifra_gosta (izračunava se iz rednog broja gosta)

Sustav za daljinsko održavanje baze podataka treba omogućiti pregled stanja baze podataka gostiju, te obradu podataka za svakog pojedinog gosta radi otklanjanja eventualnih pogrešaka ili specifičnih informacija (npr. da ne želi odgovoriti na upitnik).

6.1.1.2 Slanje e-mail poruka gostima

E-mail poruke koje se šalju gostima trebaju sadržavati kratak tekst koji sadrži prezime gosta, zajedno s titulom (ovisno o spolu), kratak tekst na njemu poznatom jeziku (jednom od pet ponuđenih) u kojem se objašnjavaju ciljevi istraživanja, te hipervezu prema odgovarajućem anketnom upitniku (osim adrese sadrži i ulazne parametre: šifru gosta i šifru jezika).
Poruke se šalju svakih tjedan dana. Ukoliko ispitanik (gost) nije ispunio upitnik, obavijest mu se šalje još jednom. Ako ni tada ne odgovori, više mu se obavijest ne šalje. Zbog toga je potrebno nakon svakog slanja e-mail poruke ažurirati u bazi podataka o gostima podatak o tome koliko puta je gost već bio kontaktiran.
Sadržaj poruke je slijedeći (tekst na hrvatskom jeziku; odnosno odgovarajući na ostalim predviđenim jezicima):

Subject: *Istraživanje zadovoljstva gostiju hotela*

Poštovani / poštovana gospodine / gospođo <prezime>!

Mi smo agencija za istraživanje tržišta iz Zagreba, Hrvatska.

Uprava gore spomenutog Hotela imenovala nas je da napravimo istraživanje zadovoljstva njihovih gostiju vezanog uz Vaš nedavni posjet istom. Njihovi podaci upućuju na to da ste pristali da Vas se može kontaktirati.

Stoga ćete na našoj web stranici pronaći link koji će Vam omogućiti da ispunite naš Upitnik o zadovoljstvu gostiju za što Vam neće trebati više od par minuta.

<adresa web upitnika>

Kao rezultat svega ovog, pribaviti ćemo odgovarajuće statističke podatke i proslijediti ih Hotelu u cilju povećanja kvalitete usluge.

Cijenimo Vaš trud i zahvaljujemo se na vašem dragocjenom vremenu.

Pozicije označene s <prezime> i <adresa web upitnika> zamjenjuju se odgovarajućim podacima, specifičnim za pojedinog gosta.

6.1.1.3 Anketiranje gostiju

E-mail poruke koje se šalju gostima trebaju sadržavati hipervezu prema odgovarajućem anketnom upitniku. Uz adresu hiperveza treba sadržavati i ulazne parametre: šifru gosta i šifru jezika. Anketiranje se sastoji od prikaza anketnog web upitnika u zadanom jeziku, gostovog ispunjavanja upitnika i upisa unesenih odgovora u bazu podataka o ispunjenim anketnim upitnicima.
Pitanja u anketnom upitniku su slijedeća (na hrvatskom jeziku; odnosno odgovarajuća na ostalim jezicima), zajedno s popratnim tekstom:

Poštovani / Poštovana,

Na većinu pitanja možete odgovoriti ocjenama od 5 do 1, gdje ocjena 5 znači izrazito zadovoljan, a 1 izrazito nezadovoljan. Ocjene između 5 i 1 mogu Vam pomoći da izrazite svoje mišljenje što preciznije.
Osvrnimo se detaljnije na Vaš nedavni boravak u Hotelu.

1. Kako biste ocijenili osoblje hotelske recepcije?					
	1	2	3	4	5
Sa stajališta učinkovitosti	☐	☐	☐	☐	☐
Sa stajališta ljubaznosti.....	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo osobljem hotelske recepcije	☐	☐	☐	☐	☐

2. Kako biste ocijenili osoblje zaduženo za održavanje čistoće?					
	1	2	3	4	5
Sa stajališta učinkovitosti	☐	☐	☐	☐	☐
Sa stajališta ljubaznosti.....	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo osobljem zaduženim za održavanje čistoće	☐	☐	☐	☐	☐

3. Kako biste ocijenili Vaš smještaj?					
	1	2	3	4	5
Izgled Vaše sobe	☐	☐	☐	☐	☐
Čistoću Vaše sobe	☐	☐	☐	☐	☐
Udobnost Vaše sobe.....	☐	☐	☐	☐	☐
Kupaonicu	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo smještajem	☐	☐	☐	☐	☐

4. Kakvo je Vaše sveukupno mišljenje o cjelokupnom prostoru?					
	1	2	3	4	5
Ulaz i predvorje	☐	☐	☐	☐	☐
Okoliš Hotela.....	☐	☐	☐	☐	☐
Plaža.....	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo prostorom	☐	☐	☐	☐	☐

5. Kako biste ocijenili osoblje zaduženo za hranu i piće?					
	1	2	3	4	5
Sa stajališta učinkovitosti	☐	☐	☐	☐	☐
Sa stajališta ljubaznosti.....	☐	☐	☐	☐	☐
6. Kako biste ocijenili ponudu hrane i pića?					
	1	2	3	4	5
Restoran - doručak	☐	☐	☐	☐	☐
Restoran - ručak	☐	☐	☐	☐	☐

Restoran - večera	☐	☐	☐	☐	☐
Lobby bar	☐	☐	☐	☐	☐
Pool bar	☐	☐	☐	☐	☐
Beach snack bar	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo uslugom vezanom uz hranu i piće	☐	☐	☐	☐	☐

7. Kako biste ocijenili osoblje u wellness i sportskim objektima?

	1	2	3	4	5
Sa stajališta učinkovitosti	☐	☐	☐	☐	☐
Sa stajališta ljubaznosti	☐	☐	☐	☐	☐

8. Kako biste ocijenili wellness i sportske objekte?

	1	2	3	4	5
Wellness centar	☐	☐	☐	☐	☐
Beauty & Health centar	☐	☐	☐	☐	☐
Fitness centar	☐	☐	☐	☐	☐
Sportska dvorana	☐	☐	☐	☐	☐
Tereni za tenis	☐	☐	☐	☐	☐
Vanjski bazen	☐	☐	☐	☐	☐
Adrenalin park	☐	☐	☐	☐	☐
Sportski centar na moru	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo wellness i sportskim objektima	☐	☐	☐	☐	☐

9. Kako biste ocijenili osoblje animacije?

	1	2	3	4	5
Sa stajališta učinkovitosti	☐	☐	☐	☐	☐
Sa stajališta ljubaznosti	☐	☐	☐	☐	☐

10. Kako biste ocijenili programe animacije?

	1	2	3	4	5
Animacija djece	☐	☐	☐	☐	☐
Dnevni program animacije	☐	☐	☐	☐	☐
Večernji program	☐	☐	☐	☐	☐
Sveukupno zadovoljstvo programom i osobljem animacije	☐	☐	☐	☐	☐

11. Sve u svemu, koliko ste zadovoljni svojim cjelokupnim boravkom u hotelu?

	1	2	3	4	5
Sveukupno	☐	☐	☐	☐	☐

Zašto?

_____

_____

12. Odgovara li cijena kvaliteti usluge?

Da ..... ☐
Ne ..... ☐

13. Biste li nas preporučili svojim prijateljima?

Da ..... ☐
Ne ..... ☐

14. Želite li dodati bilo kakav komentar vezan uz Vaš nedavni posjet?

_____

_____

15. Slažete li se ili ne da Vaše odgovore proslijedimo Upravi Hotela?

Slažem se ..... ☐
Ne slažem se – želim ostati anoniman..... ☐

Prema tome, ispitanici na većinu pitanja odgovaraju izborom jedne od 5 ponuđenih mogućnosti, na dva pitanja postoje dvije mogućnosti odgovora, te na još dva se daje tekstualni odgovor. Nije obavezno odgovoriti na sva pitanja.

6.1.1.4 Prikaz i obrada rezultata ankete

Rezultate anketiranja potrebno je prikazati grafički pomoću dijagrama sa stupcima, te tekstualno u obliku koji se može učitati u tablični kalkulator, kao što je Excel. Obrada rezultata koja je predviđena u okviru sustava anketiranja obuhvaća izračunavanje i prikaz djelomičnih rezultata anketiranja, ovisno o slijedećim karakteristikama ispitanika:

- starost (nekoliko ponuđenih raspona godina),
- spol,
- segment gostiju,
- nacionalnost i
- redni broj tjedna

Na taj način omogućeno je, osim prikaza sveukupnih rezultata anketiranja, filtriranje odgovora i prikaz rezultata ispitanika prema navedenim karakteristikama.

6.1.2 Generativni razvoj sustava

Pojedini dijelovi sustava za anketiranje gostiju hotela mogu se realizirati korištenjem generatora obrađenih u poglavlju 5, bez potreba za modifikacijama generatora, dok su za ostale dijelove potrebne određene modifikacije generatora. U toku izvođenja projekta može doći do određenih promjena u projektnom zadatku. Generativni razvoj aplikacija treba omogućiti jednostavne promjene u sustavu, bez obzira na njihove implikacije na programski kod, koji može biti disperziran na veći broj programskih modula.

6.1.2.1 Strojni i programski elementi sustava za anketiranje gostiju

Sustav za anketiranje gostiju hotela zamišljen je kao web sustav s dva osnovna dijela:

- administracijski dio - obuhvaća održavanje baze podataka o gostima, slanje e-mail poruka, te prikaz i obradu rezultata ankete i
- korisnički dio - anketni upitnik

Administracijski dio treba biti zaštićen odgovarajućom zaporkom, a korisničkom dijelu pristupa se s podatkom o šifri gosta, koja mora postojati u bazi podataka o gostima.

Web poslužitelj:

- operacijski sustav UNIX (ili srodan) s instaliranim Apache web poslužiteljem
- interpreter za Perl
- MySQL baza podataka
- instalirani programski moduli za Perl:
    - DBI - modul za povezivanje s bazom podataka
    - SENDMAIL - modul za slanje e-mail poruka.

6.1.2.2 Primijenjeni generatori

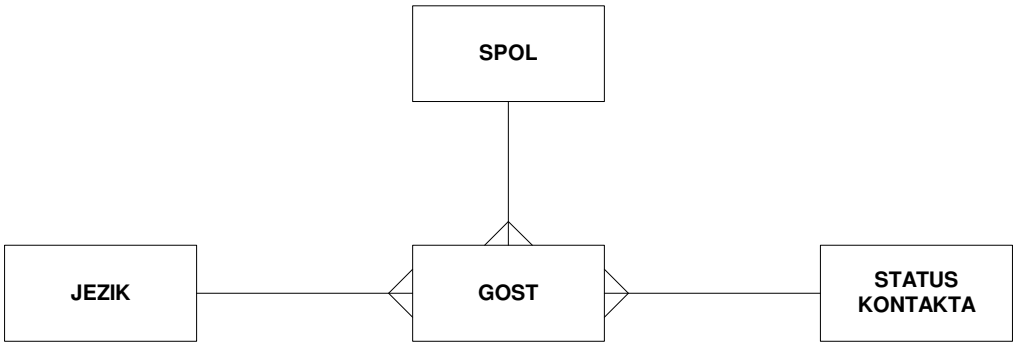
U realizaciji sustava za anketiranje gostiju hotela primijenjeni su slijedeći generatori, detaljnije obrađeni u okviru poglavlja 5:

- **generator aplikacija za daljinsko održavanje baza podataka** upotrijebljen je za generiranje aplikacije za održavanje baze podataka o gostima, osim dijela za učitavanje podataka iz datoteke u CSV formatu, te baze predložaka za aplikaciju generiranu pomoću generatora korisničkih web aplikacija,
- **generator korisničkih web aplikacija** za učitavanje podataka o gostima iz datoteke u CSV formatu, slanje e-mail poruka gostima i prikaz anketnog upitnika s tekstom u zadanom jeziku i
- **generator anketnih upitnika** za generiranje anketnih upitnika, uključujući prikaz i obradu rezultata.

6.1.2.3 Održavanje baze podataka o gostima i baze predložaka

Održavanje baze podataka o gostima, te baze predložaka realizirano je većim dijelom pomoću generatora aplikacija za daljinsko održavanje baza podataka, osim dijela za učitavanje podataka iz datoteke u CSV formatu (poglavlje 6.1.2.4).

ER model baze podataka o gostima možemo vidjeti na slici 6.2:



Slika 6.2: ER model baze podataka o gostima

Baza predložaka sadrži tablice u kojima se nalaze:

- predlošci e-mail poruka za poruke na 5 jezika,
- predlošci anketnih upitnika na 5 jezika i
- predlošci web stranica za aplikaciju koja učitava podatke iz datoteke u CSV formatu, te šalje e-mail poruke

Odgovarajući opis aplikacije za održavanje baze podataka o gostima i baze predložaka je slijedeći:

```
konekcija:$DBH=DBI->connect
("dbi:mysql:dbname=acc;host=localhost","acc","ac4tp6u4");

naslov:gosti
tablica:ac_gosti4
polje_broj:redni_broj_gosta
polje_tekst:prezime
polje_tekst:ime
polje_combo:spol,tablica:ac_spol,kljuc:r_broj,polje:spol_gosta,vrijednost:gosti
polje_tekst:e_mail
polje_tekst:adresa
polje_tekst:telefon
polje_tekst:fax
polje_tekst:nacionalnost
polje_combo:jezik_za_kontakt,tablica:ac_jezik,kljuc:r_broj,polje:jezik,vrijednost:gosti
polje_tekst:segment_gostiju
polje_combo:kontaktiran,tablica:ac_kontakt,kljuc:r_broj,polje:kontakt,vrijednost:gosti
polje_tekst:datum_rodjenja
polje_tekst:dolazak
polje_tekst:odlazak
polje_broj:redni_broj_tjedna
polje_tekst:sifra_gosta

naslov:poruke
tablica:ac_poruke4
polje_combo:jezik,tablica:ac_jezik,kljuc:r_broj,polje:jezik,vrijednost:gosti
polje_tekst:subject
polje_tekst:zaglavlje_m
polje_tekst:zaglavlje_z
polje_memo:predlozak
polje_memo:tekst1
polje_memo:tekst2
polje_memo:tekst3

naslov:anketni obrasci
tablica:ac_obrasci
polje_combo:jezik,tablica:ac_jezik,kljuc:r_broj,polje:jezik,vrijednost:gosti
polje_memo:obrazac

naslov:predlosci
tablica:ac_predlosci
polje_tekst:naziv_predloska
polje_memo:predlozak

naslov:spol
tablica:ac_spol
polje_tekst:spol_gosta

naslov:jezik
tablica:ac_jezik
polje_tekst:jezik

naslov:status kontakta
tablica:ac_kontakt
polje_tekst:kontakt
```

6.1.2.4 Slanje e-mail poruka gostima i učitavanje podataka o gostima

Slanje e-mail poruka gostima i učitavanje podataka o gostima iz datoteke u CSV formatu riješeno je pomoću generatora korisničkih web aplikacija. Odgovarajući opis aplikacije je slijedeći:

```
konekcija:$DBH=DBI->connect
("dbi:mysql:dbname=accent;host=localhost","accent","ac4tp6u4");
globalna:vrsta,4

predlozak_stranice:select predlozak from ac_predlosci where r_broj=$vrsta
++++++++++++++++++++++++++++++++++++++++
#obavijesti#
sql:select ac_gosti4.r_broj, spol, prezime, ime, e_mail, kontaktiran, subject, predlozak,
zaglavlje_m,zaglavlje_z,tekst1,tekst2,tekst3,jezik_za_kontakt from ac_gosti4,ac_poruke4
where ac_gosti4.jezik_za_kontakt=ac_poruke4.jezik and kontaktiran < 3
b_prikaza:100000
polje:r_broj
polje:spol
polje:prezime
polje:ime
polje:e_mail
polje:kontaktiran
polje:subject
polje:predlozak
polje:zaglavlje_m
polje:zaglavlje_z
polje:tekst1
polje:tekst2
polje:tekst3
polje:jezik_za_kontakt
i:$contattat="contattato";
i:$tekst_poruke=$predlozak;
i:$zaglavlje=$zaglavlje_m;
i:if ($spol == 2){$zaglavlje=$zaglavlje_z;$contattat="contattata";}
i:$tekst=$tekst1;
i:if ($kontaktiran == 2){$tekst=$tekst2;}
i:if ($kontaktiran == 3){$tekst=$tekst3;}
i:$tekst=~ s/contattato\/a/$contattat/;
i:$tekst_poruke =~ s/#tekst#/$tekst/;
i:$tekst_poruke =~ s/#zaglavlje#/$zaglavlje/;
i:$sifra_gosta=$r_broj*57+16534;
i:$adresa_slike="http://www.accent.hr/slike/accent.jpg";
i:$tekst_poruke =~ s/#slika#/$adresa_slike/;
i:$crta="http://www.accent.hr/slike/crta.jpg";
i:$tekst_poruke =~ s/#crta#/$crta/;
i:$tekst_poruke =~ s/#crta#/$crta/;
i:$pozadina="http://www.accent.hr/slike/text_ponuda.jpg";
i:$tekst_poruke =~ s/#pozadina#/$pozadina/;
i:$tekst_poruke =~ s/#gost#/$prezime/;
i:$adresa="http://www.accent.hr/cgi-
bin/upitnik/ulaz.cgi?jk=$jezik_za_kontakt&id=$sifra_gosta";
i:$tekst_poruke =~ s/#adresa#/$adresa/;
i:$tekst_poruke =~ s/#adresa#/$adresa/;
i:$zamjene="$zamjene$prezime $ime - $e_mail<br>";
i:$ime = "$prezime $ime";
```

```
i:$primatelji ="\"$ime\" <$e_mail>";
i:$posiljatelj = "\"$ime\" <$e_mail>";
i:$subject =~ s/ž/\=BE/;
i:$subject =~ s/-/\=2D/;
i:$subject =~ s/š/\=B9/;
i:$subject =~ s/Š/\=A9/;
i:$primatelji =~ s/ž/\=BE/;
i:$primatelji =~ s/š/\=B9/;
i:$primatelji =~ s/Š/\=A9/;
i:$primatelji =~ s/Ž/\=AE/;
i:open(SENDMAIL, "|/usr/lib/sendmail -t") or die "Pogreška: ne mogu poslati mail! $!\n";
i:$poruka="From:Accent - Zagreb <info\@accent.hr>";
i:$poruka="$poruka\nMIME-Version: 1.0";
i:$poruka="$poruka\nContent-Type: text\/html\;   charset=\"windows-1250\"";
i:$poruka="$poruka\nContent-Transfer-Encoding: 8bit";
i:$poruka="$poruka\nTo: =?iso-8859-2?Q?$primatelji?=";
i:#$poruka="$poruka\ncc: Danijel <darados\@barok.foi.hr>";
i:$poruka="$poruka\nSubject: =?iso-8859-2?Q?$subject?=";
i:$poruka="$poruka\n \n$tekst_poruke";
i:print SENDMAIL $poruka;
i:#print SENDMAIL "$sadrzaj\n\n";
i:print SENDMAIL "\n\n";
i:print SENDMAIL "";
i:close(SENDMAIL)     or warn "Pogreška kod zatvaranja maila!";
i:if ($kontaktiran < 4){$kontaktiran++;}
i:$upit="update ac_gosti4 set sifra_gosta='$sifra_gosta', kontaktiran=$kontaktiran where
r_broj=$r_broj";
i:my $sth=$DBH->prepare("$upit");
i:if ($sth->execute){} else {print "Database error!";}
----------------------------------------
#upload#
sql:select prezime from ac_gosti4
b_prikaza:1
polje:prezime
p:$podaci_o_gostima=$query->param('podaci_o_gostima');
p:(@redovi)=split(/\n/,$podaci_o_gostima);
p:$br=1;
p:foreach $line (@redovi){
p:if ($br > 2){
p:#print "$line *** <br>";
p:($a,$a,$a,$a,$a,$a,$a,$a,$redni_broj_gosta,$a,$a,$a,$ime,$prezime,$dolazak,$odlazak,$a,$a,
$jezik1,$a,$a,$a,$a,$nacionalnost,$segment_gostiju,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a
,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$a,$datum_rodjenja,$spol1,$a,$a,$a,$a,$a,$a,$a,$
a,$a,$ulica,$post_broj,$mjesto,$a,$a,$a,$a,$a,$a,$a,$telefon,$fax,$e_mail,$a,$a,$a,$a,$a,$a
,$a,)=split(/\;/,$line);
p:$jezik=2;
p:if ($jezik1 eq "hrv") {$jezik="1";}
p:if ($jezik1 eq "eng") {$jezik="2";}
p:if ($jezik1 eq "deu") {$jezik="3";}
p:if ($jezik1 eq "ita") {$jezik="4";}
p:if ($jezik1 eq "slo") {$jezik="5";}
p:$jezik_za_kontakt=$jezik;
p:$spol="1";
p:if ($spol1 ne "M") {$spol="2"}
p:$adresa="$ulica $post_broj $mjesto";
p:$kontaktiran=1;
p:$sifra_gosta="0";
p:$datum=$odlazak;
p:($dan,$mjesec,$godina)=split(/\./,$datum);$dana=0;if ($godina%4==0){$dana=1;}
p:if ($mjesec > 1){$dana=$dana+31;}
```

```
p:if ($mjesec > 2){$dana=$dana+28;}
p:if ($mjesec > 3){$dana=$dana+31;}
p:if ($mjesec > 4){$dana=$dana+30;}
p:if ($mjesec > 5){$dana=$dana+31;}
p:if ($mjesec > 6){$dana=$dana+30;}
p:if ($mjesec > 7){$dana=$dana+31;}
p:if ($mjesec > 8){$dana=$dana+31;}
p:if ($mjesec > 9){$dana=$dana+30;}
p:if ($mjesec > 10){$dana=$dana+31;}
p:if ($mjesec > 11){$dana=$dana+30;}
p:$dana=$dana+$dan-1;
p:$redni_broj_tjedna=int($dana / 7 + 1);
p:#print "$dan,$mjesec,$godina|$dana|$redni_broj_tjedna<br>";
p:if (length($e_mail) > 1){
p:print "$redni_broj_gosta,$ime,
$prezime,$dolazak,$odlazak,$jezik_za_kontakt,$nacionalnost,$datum_rodjenja,$spol,tel:$tele
fon,fax:$fax,mail:$e_mail,$adresa,$segment_gostiju,$redni_broj_tjedna<br>";
p:$upit="insert into ac_gosti4 (redni_broj_gosta, prezime, ime, spol, e_mail, adresa,
telefon,fax,nacionalnost,jezik_za_kontakt,segment_gostiju,kontaktiran,datum_rodjenja,dolaza
k,odlazak,redni_broj_tjedna,sifra_gosta) values ($redni_broj_gosta, '$prezime', '$ime', $spol,
'$e_mail', '$adresa', '$telefon', '$fax', '$nacionalnost', $jezik_za_kontakt, '$segment_gostiju',
$kontaktiran, '$datum_rodjenja','$dolazak','$odlazak',$redni_broj_tjedna,'$sifra_gosta')";
p:#print "$upit<br>";
p:my $sth=$DBH->prepare("$upit");
p:if ($sth->execute){} else {print "Database error!";}
p:}
p:}
p:$br++;}
p:$zamjene="Podaci su uspješno prenešeni!";
----------------------------------------
```

Predložak stranice učitava se pomoću SQL upita unutar oznake 'predlozak_stranice'. Oznaka '#obavijesti#' odnosi se na programski kod za slanje e-mail poruka gostima hotela. Pomoću odgovarajućeg SQL upita učitaju se podaci iz tablice gostiju za one goste koji su do sada najviše jednom kontaktirani, te iz tablice s porukama na različitim jezicima. U tekstu poruke odgovarajuće oznake zamjenjuju se prezimenom gosta, odnosno adresom web upitnika.
Oznaka '#upload#' označava programski kod koji se odnosi na učitavanje podataka o gostima iz datoteke u CSV formatu. Koriste se mogućnosti jezika Perl u obradi znakovnih nizova da bi se izdvojili podaci koji se upisuju u bazu podataka o gostima.

6.1.2.5 Anketiranje gostiju

Anketiranje gostiju realizirano je pomoću dva generatora: generatora korisničkih web aplikacija i generatora anketnih upitnika. Generator anketnih upitnika generira anketni obrazac, no u sustavu za anketiranje gostiju hotela potrebni su anketni obrasci na pet različitih jezika. Zbog toga je potrebno generirani anketni upitnik grafički obraditi i prevesti u nekom od programa za obradu HTML dokumenata, kao što je FrontPage. Obrađene anketne upitnike treba smjestiti u odgovarajuću tablicu u bazi podataka, odakle ih učitava i prikazuje odgovarajuća CGI skripta, izrađena pomoću generatora korisničkih web aplikacija. Odgovarajući opis aplikacije je slijedeći:

```
konekcija:$DBH=DBI->connect ("dbi:Pg:dbname=darados;host=alf.cat.foi.hr","darados","");
globalna:jk,0
globalna:id,0
```

```
predlozak_stranice:select obrazac from ac_obrasci where jezik='$jk'
++++++++++++++++++++++++++++++++++++++++
#provjera1#
sql:select sifra_gosta from ac_gosti4 where sifra_gosta='$id'
b_prikaza:10000
polje:sifra_gosta
z:$brojac=0;
i:$brojac++;
p:if ($brojac == 0){
p:print "<b>Error!</b>";
p:exit;
p:}
----------------------------------------
#provjera2#
sql:select sifra_gosta from ac_upitnik2 where sifra_gosta='$id'
b_prikaza:10000
polje:sifra_gosta
z:$brojac=0;
i:$brojac++;
p:if ($brojac > 0){
p:print "<b>Thank You, we already have Your results!</b>";
p:exit;
p:}
----------------------------------------
```

Možemo vidjeti da se koriste dva ulazna parametra, koji se odnose na jezik i na šifru gosta. Ovisno o jeziku, učitava se odgovarajući predložak stranice, koji sadrži anketni upitnik. Oznaka '#provjera1#' odnosi se na provjeru postojanja šifre gosta u bazi podataka o gostima, a '#provjera2#' na provjeru da li šifra gosta postoji u tablici ispunjenih anketnih upitnika (to bi značilo da je gost već ispunio anketni upitnik).

Generator anketnih upitnika koristi slijedeći opis aplikacije:

```
tablica:ac_upitnik2
naslov:Istraživanje zadovoljstva gostiju hotela
pitanje_J:Kako biste ocijenili osoblje hotelske recepcije?\nSa stajališta učinkovitosti
5
4
3
2
1
pitanje_J:Sa stajališta ljubaznosti
5
4
3
2
1
pitanje_J:Sveukupno zadovoljstvo osobljem hotelske recepcije
5
4
3
2
1
pitanje_J:Kako biste ocijenili osoblje zaduženo za održavanje čistoće?\nSa stajališta učinkovitosti
5
4
3
2
1
pitanje_J:Sa stajališta ljubaznosti
5
4
```

3
2
1
pitanje_J:*Sveukupno zadovoljstvo osobljem zaduženim za održavanje čistoće*
5
4
3
2
1
pitanje_J:*Kako biste ocijenili Vaš smještaj?\nIzgled Vaše sobe*
5
4
3
2
1
pitanje_J:*Čistoću Vaše sobe*
5
4
3
2
1
pitanje_J:*Udobnost Vaše sobe*
5
4
3
2
1
pitanje_J:*Kupaonicu*
5
4
3
2
1
pitanje_J:*Sveukupno zadovoljstvo smještajem*
5
4
3
2
1
pitanje_J:*Kakvo je Vaše sveukupno mišljenje o cjelokupnom prostoru?\nUlaz i predvorje*
5
4
3
2
1
pitanje_J:*Okoliš Hotela*
5
4
3
2
1
pitanje_J:*Plaža*
5
4
3
2
1
pitanje_J:*Sveukupno zadovoljstvo prostorom*
5
4
3
2
1
pitanje_J:*Kako biste ocijenili osoblje zaduženo za hranu i piće?\nSa stajališta učinkovitosti*
5
4
3
2
1
pitanje_J:*Sa stajališta ljubaznosti*
5
4

3
2
1
pitanje_J:*Kako biste ocijenili ponudu hrane i pića?\nRestoran - doručak*
5
4
3
2
1
NA
pitanje_J:*Restoran - ručak*
5
4
3
2
1
NA
pitanje_J:*Restoran - večera*
5
4
3
2
1
NA
pitanje_J:*Lobby bar*
5
4
3
2
1
NA
pitanje_J:*Pool bar*
5
4
3
2
1
NA
pitanje_J:*Beach snack bar*
5
4
3
2
1
NA
pitanje_J:*Sveukupno zadovoljstvo uslugom vezanom uz hranu i piće*
5
4
3
2
1
NA
pitanje_J:*Kako biste ocijenili osoblje u wellness i sportskim objektima?\nSa stajališta učinkovitosti*
5
4
3
2
1
NA
pitanje_J:*Sa stajališta ljubaznosti*
5
4
3
2
1
NA
pitanje_J:*Kako biste ocijenili wellness i sportske objekte?\nWellness centar*
5
4
3
2
1

NA
pitanje_J:*Beauty & Health centar*
5
4
3
2
1
NA
pitanje_J:*Fitness centar*
5
4
3
2
1
NA
pitanje_J:*Sportska dvorana*
5
4
3
2
1
NA
pitanje_J:*Tereni za tenis*
5
4
3
2
1
NA
pitanje_J:*Vanjski bazen*
5
4
3
2
1
NA
pitanje_J:*Adrenalin park*
5
4
3
2
1
NA
pitanje_J:*Sportski centar na moru*
5
4
3
2
1
NA
pitanje_J:*Sveukupno zadovoljstvo wellness i sportskim objektima*
5
4
3
2
1
NA
pitanje_J:*Kako biste ocijenili osoblje animacije?\nSa stajališta učinkovitosti*
5
4
3
2
1
NA
pitanje_J:*Sa stajališta ljubaznosti*
5
4
3
2
1
NA
pitanje_J:*Kako biste ocijenili programe animacije?\nAnimacija djece*
5

4
3
2
1
NA
pitanje_J:*Dnevni program animacije*
5
4
3
2
1
NA
pitanje_J:*Večernji program*
5
4
3
2
1
NA
pitanje_J:*Sveukupno zadovoljstvo programom i osobljem animacije*
5
4
3
2
1
NA
pitanje_J:*Sve u svemu, koliko ste zadovoljni svojim cjelokupnim boravkom u hotelu?\nSveukupno*
5
4
3
2
1
pitanje_T:*Zašto?*
pitanje_J:*Odgovara li cijena kvaliteti usluge?*
da
ne
pitanje_J:*Biste li nas preporučili svojim prijateljima?*
da
ne
pitanje_T:*Želite li dodati bilo kakav komentar vezan uz Vaš nedavni posjet?*
pitanje_J:*Slažete li se ili ne da Vaše odgovore proslijedimo Upravi Hotela?*
Slažem se
Ne slažem se - želim ostati anoniman
polja:
recepcija1
recepcija2
recepcija3
cistoca1
cistoca2
cistoca3
smjestaj1
smjestaj2
smjestaj3
smjestaj4
smjestaj5
prostor1
prostor2
prostor3
prostor4
osoblje_za_hranu_i_pice1
osoblje_za_hranu_i_pice2
ponuda_hrane_i_pica1
ponuda_hrane_i_pica2
ponuda_hrane_i_pica3
ponuda_hrane_i_pica4
ponuda_hrane_i_pica5
ponuda_hrane_i_pica6
ponuda_hrane_i_pica7
wellness_i_sport_osoblje1
wellness_i_sport_osoblje2
wellness_i_sport_objekti1
wellness_i_sport_objekti2

wellness_i_sport_objekti3
wellness_i_sport_objekti4
wellness_i_sport_objekti5
wellness_i_sport_objekti6
wellness_i_sport_objekti7
wellness_i_sport_objekti8
wellness_i_sport_objekti9
osoblje_animacije1
osoblje_animacije2
program_animacije1
program_animacije2
program_animacije3
program_animacije4
ukupno_zadovoljstvo
napomena1
cijena_kvaliteta
preporuka
napomena2
proslijediti_upravi
parcijalno:

Može se vidjeti da se anketni upitnik sastoji uglavnom od pitanja s jednom mogućim odgovorom, osim dva pitanja, gdje se traži tekstualni odgovor. Neka pitanja imaju ponuđen odgovor NA (eng. Not Applicable - nije primjenjivo), koja se odnose na one sadržaje koje gost nije koristio, pa ih ne može ni ocijeniti.

6.1.2.6 Prikaz i obrada rezultata ankete

Generator anketnih upitnika omogućuje, među ostalim i prikaz rezultata anketnih upitnika u grafičkom i tekstualnom obliku. To zadovoljava za prikaz ukupnih rezultata ankete u grafičkom obliku, ali u slučaju sustava za anketiranje gostiju hotela ne i za prikaz djelomičnih rezultata ankete u grafičkom obliku, te za prikaz rezultata u tekstualnom obliku. Za prikaz djelomičnih rezultata ankete predviđen je u opisu aplikacije odjeljak 'parcijalno'. Međutim, u tom odjeljku moguće je zadavati samo one kriterije koji se odnose na odgovore iz upitnika, a traženi kriteriji se nalaze u bazi podataka o gostima. Prema tome, potrebne su odgovarajuće modifikacije generatora anketnih upitnika, kako bi se omogućio grafički prikaz rezultata prema slijedećim kriterijima:

- starost (nekoliko ponuđenih raspona godina),
- spol,
- segment gostiju,
- nacionalnost i
- redni broj tjedna.

Također, u tekstualnim rezultatima ankete potrebno je uključiti te podatke (kriterije za prikaz u grafičkom obliku) u ispis.

6.1.2.6.1 Modifikacije generatora anketnih upitnika

Da bi se omogućio prikaz rezultata u grafičkom i tekstualnom obliku prema kriterijima navedenim u poglavlju 6.1.2.6 potrebno je izvršiti slijedeće izmjene u odgovarajućim metaskriptama za prikaz rezultata ankete ('GRAFIČKI-PRIKAZ' i 'TEKSTUALNI-PRIKAZ'), vidljivim na odgovarajućem dijagramu metaskripata (slika 5.16):

- modificirati SQL upit kojim se čitaju rezultati anketnih upitnika tako da se uključe polja iz tablice gostiju (veza između tih tablica je polje 'sifra_gosta').

*$sql="select * from #tablica# $dod";*

treba promijeniti u:

$sql="select datum_rodjenja, prezime, ime,spol, redni_broj_tjedna, segment_gostiju, nacionalnost, #tablica#. from ac_gosti4, #tablica# where ac_gosti4.sifra_gosta=#tablica#.sifra_gosta $dod order by #tablica#.r_broj;*

- modificirati izraz za čitanje skupa zapisa iz baze podataka, tako da uključi polja iz tablice gostiju:

($r_broj#polja,$id2)=@$row;

treba promijeniti u:

($datum_rodjenja, $prezime, $ime, $spol, $redni_broj_tjedna, $segment_gostiju, $nacionalnost, $r_broj#polja#, $id2)=@$row;

- modificirati ispis tekstualnih rezultata, tako da uključi polja iz tablice gostiju**:**

*print "$r_broj,$sifra_gosta#polja#,$id2
";*

treba promijeniti u:

*print "$datum_rodjenja, $prezime, $ime, $spol, $redni_broj_tjedna, $segment_gostiju, $nacionalnost, $r_broj,$sifra_gosta#polja#,$id2
";*

- uključiti zadane kriterije u prikaz rezultata ankete u grafičkom obliku. Potrebno je u metaskriptu 'GRAFIČKI-PRIKAZ' dodati slijedeće instrukcije za rad s parametrima koji se odnose na prikaz djelomičnih rezultata (varijabla '$dod' sadrži dio SQL upita koji treba omogućiti filtriranje rezultata ankete):

```
$spol=$query->param('spol');
$tjedan=$query->param('tjedan');
$segment=$query->param('segment');
$nacionalnost=$query->param('nacionalnost');
$dob=$query->param('dob');

if ($spol ne ""){$dod="$dod and spol=$spol";}
if ($tjedan ne ""){$dod="$dod and redni_broj_tjedna=$tjedan";}
if ($segment ne ""){$dod="$dod and segment_gostiju='$segment'";}
if ($nacionalnost ne ""){$dod="$dod and nacionalnost='$nacionalnost'";}
```

Ove modifikacije na razini pojedinih metaskripata omogućile su prilagodbu generatora anketnih upitnika potrebama projekta anketiranja gostiju hotela.

6.2 Izrađeni sustav za anketiranje gostiju hotela

Sustav za anketiranje gostiju hotela izrađen je korištenjem tri generatora. U slučaju generatora aplikacija za daljinsko održavanje baze podataka, te generatora korisničkih web aplikacija, sve modifikacije odnosile su se na opis aplikacije, dok je u

slučaju generatora anketnih upitnika trebalo modificirati i metaskripte. Gotov sustav ima slijedeće funkcionalnosti:

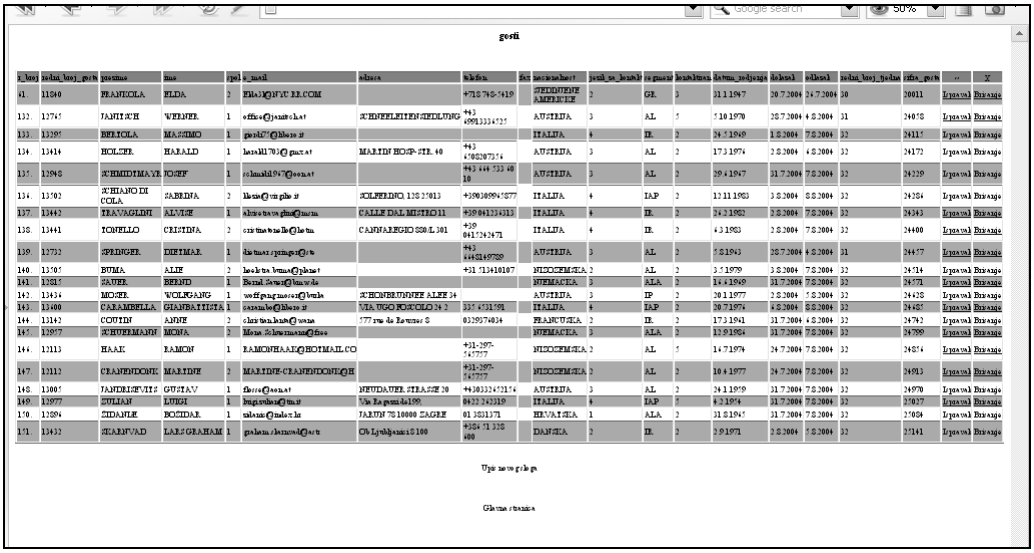
- održavanje baza podataka o gostima i baze predložaka,
- učitavanje podataka o gostima i slanje e-mail poruka,
- anketiranje gostiju i
- prikaz i obrada rezultata ankete.

6.2.1 Održavanje baze podataka o gostima i baze predložaka

Održavanje baza podataka o gostima i baze predložaka koristi se kao jedna aplikacija, čiji glavni izbornik možemo vidjeti na slici 6.3 .

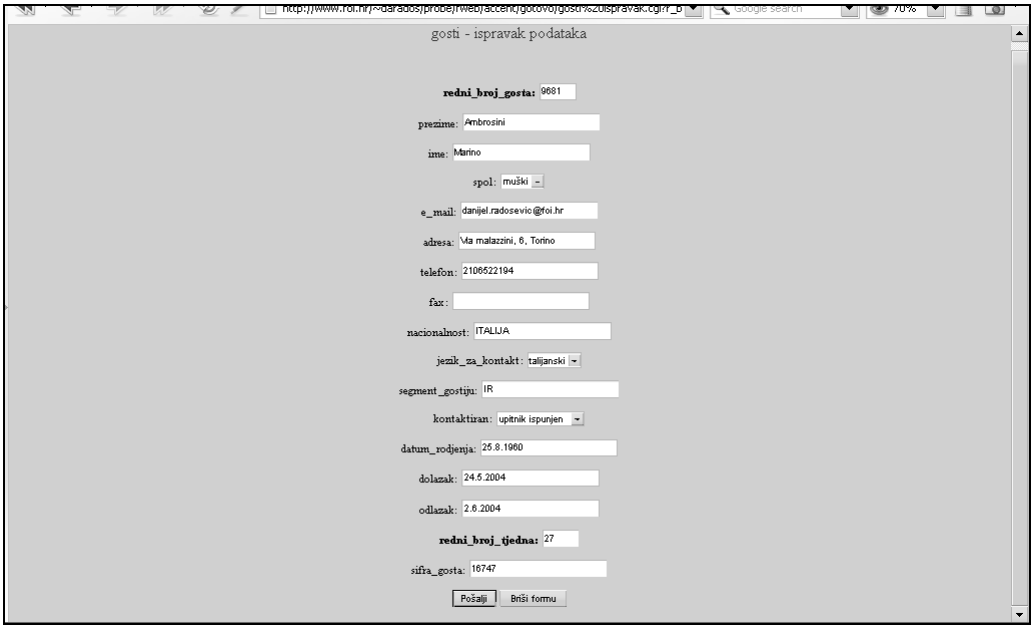
Slika 6.3. Glavni izbornik aplikacije za održavanje baze podataka

Primjer ispisa sadržaja baze podataka gostiju može se vidjeti na slici 6.4:



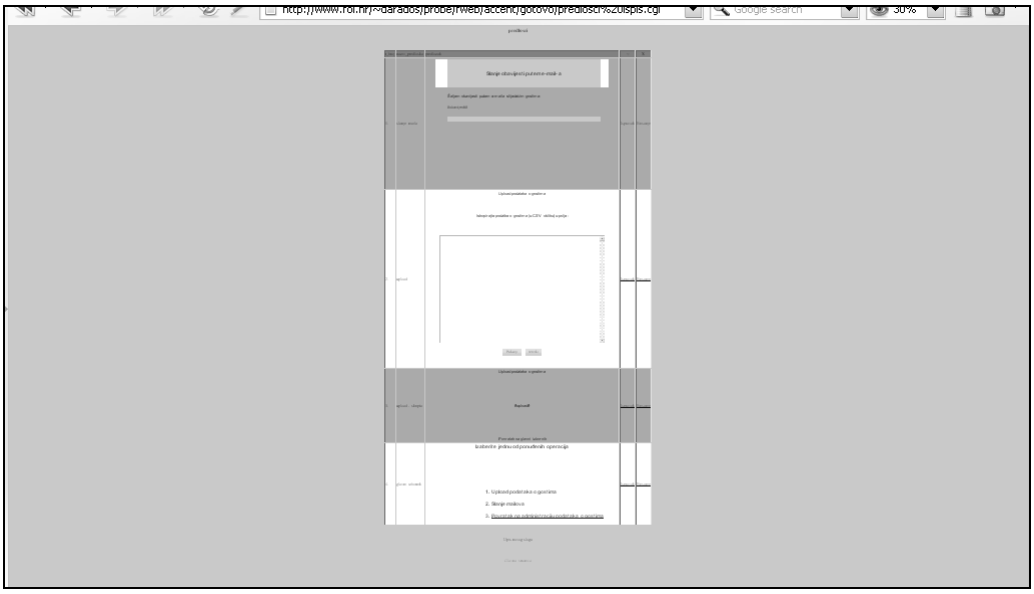
Slika 6.4: Primjer ispisa sadržaja baze podataka gostiju

Odnosno, ispravak pojedinog zapisa u tablici gostiju moguć je pomoću odgovarajućeg obrasca (slika 6.5):



Slika 6.5: Ispravak pojedinog zapisa u tablici gostiju

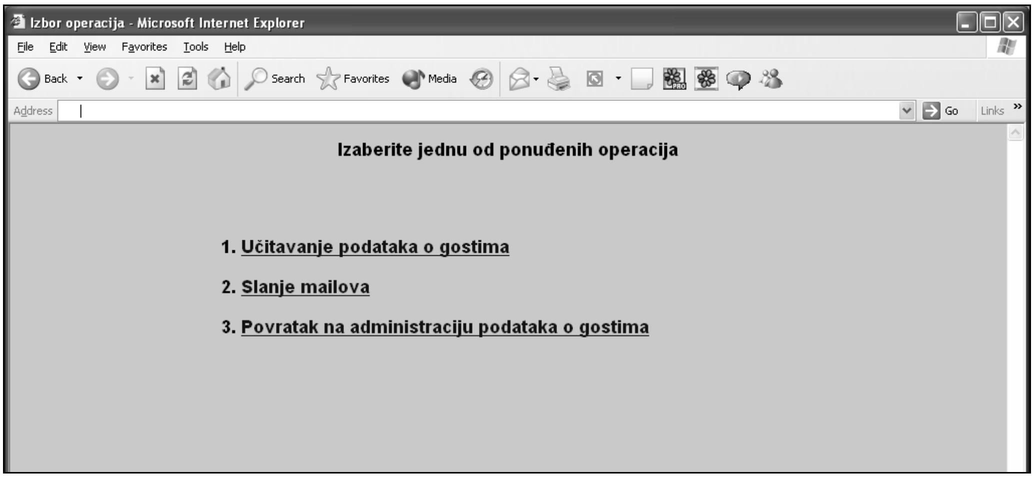
Primjer ispisa sadržaja tablice predložaka stranica koje koristi aplikacija za učitavanje podataka o gostima i slanje e-mail poruka vidljiv je na slici 6.6:



Slika 6.5: Primjer ispisa sadržaja tablice predložaka stranica

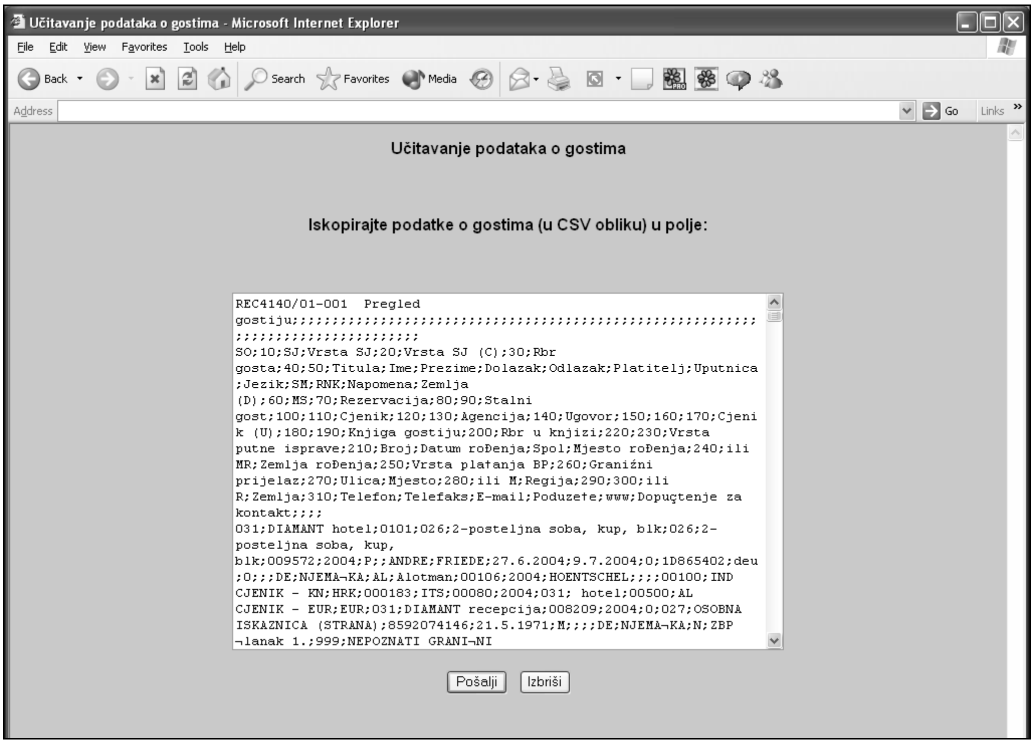
6.2.2 Učitavanje podataka o gostima i slanje e-mail poruka

Za učitavanje podataka o gostima i slanje e-mail poruka koristi se kao jedna aplikacija, generirana pomoću generatora korisničkih web aplikacija. Izgled izbornika te aplikacije vidi se na slici 6.6:



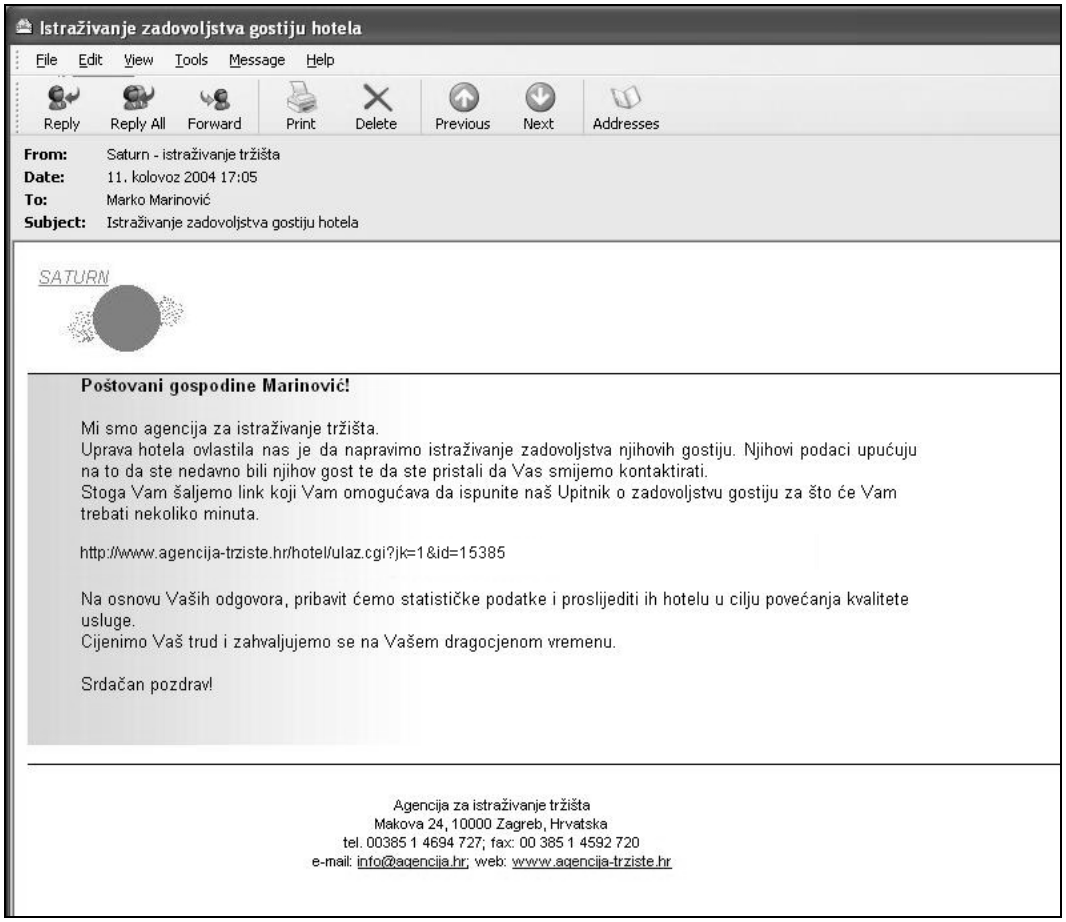
Slika 6.6: Izbornik aplikacije za učitavanje podataka o gostima i slanje e-mail poruka

Učitavanje podataka o gostima izvodi se pomoću odgovarajućeg obrasca za unos podataka (slika 6.7). Ulazni podaci su u CSV formatu.



Slika 6.7: Obrazac za učitavanje podataka o gostima

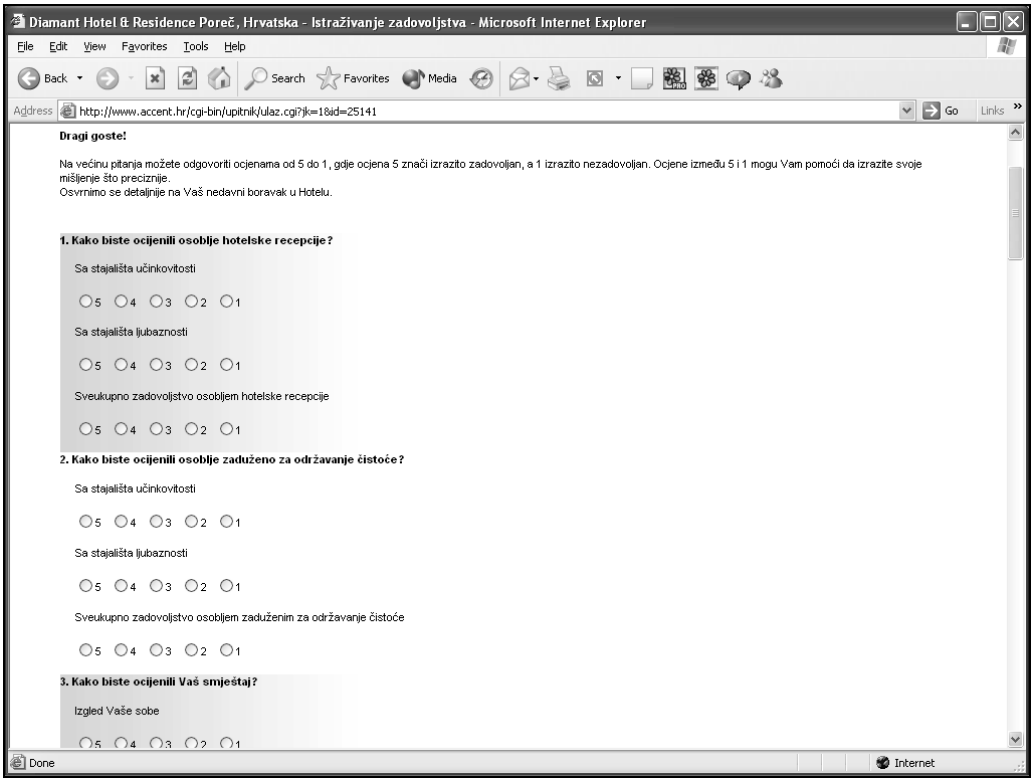
E-mail poruke koje se šalju gostima imaju slijedeći izgled (slika 6.8), s tim da se tekst mijenja ovisno o jeziku, spolu i prezimenu gosta:



Slika 6.8: Primjer e-mail poruke gostu

6.2.3 Anketiranje gostiju

E-mail poruke koje se šalju gostima hotela sadrže adresu anketnog web upitnika, koji postoji u pet varijanti, ovisno o jeziku. Također, svaki gost ispitanik može anketni upitnik ispuniti samo jednom. Nije obavezno odgovoriti na sva pitanja. Izgled anketnog upitnika vidljiv je na slici 6.9.

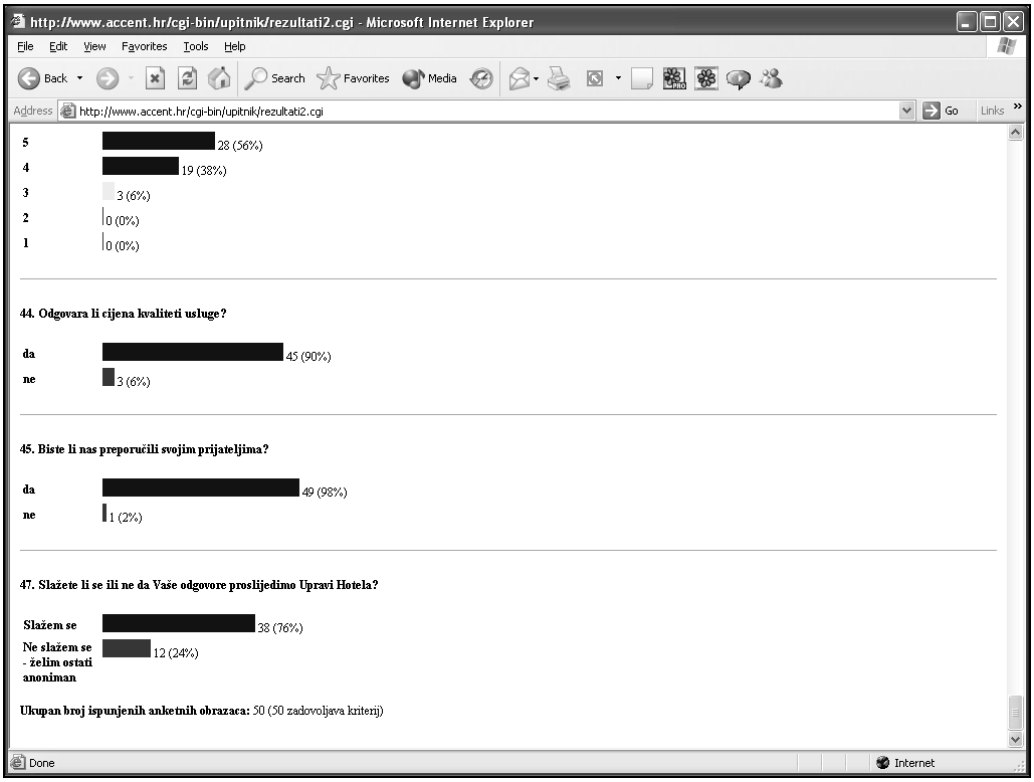


Slika 6.9: Izgled anketnog upitnika (na hrvatskom jeziku)

Većina pitanja u upitniku je s jednim mogućim odgovorom, a na dva pitanja je predviđen tekstualni odgovor.

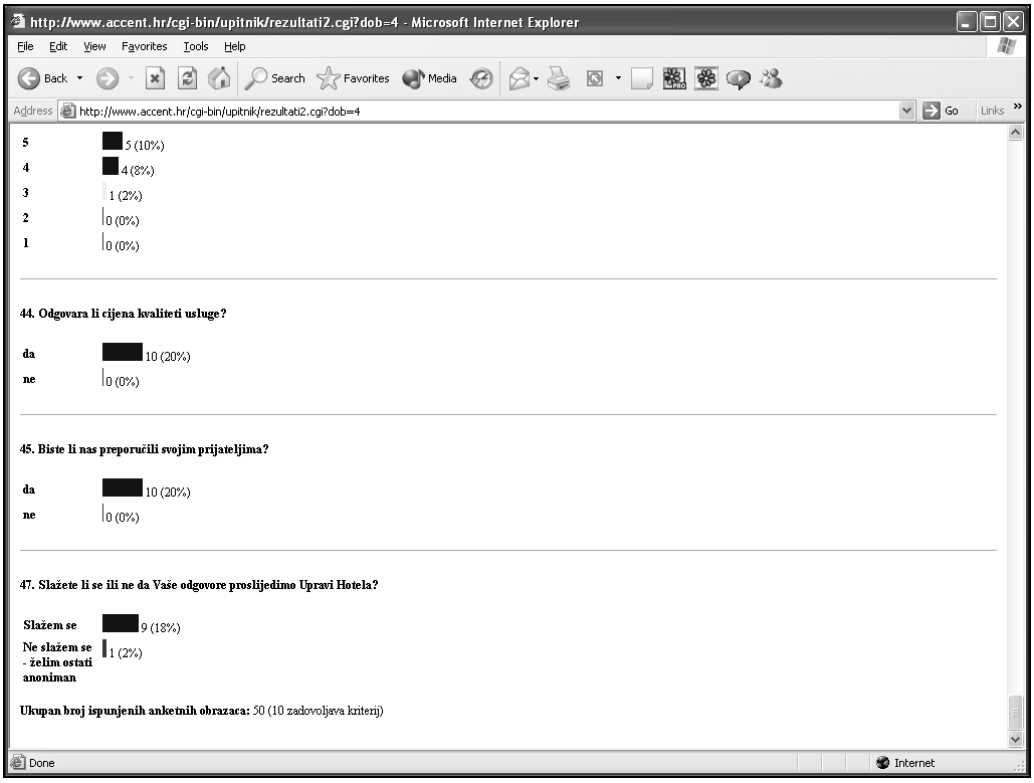
6.2.4 Prikaz i obrada rezultata ankete

Prikaz i obrada rezultata ankete mogući su u grafičkom i tekstualnom obliku. Prikaz ukupnih rezultata u grafičkom obliku vidljiv je na slici 6.10:



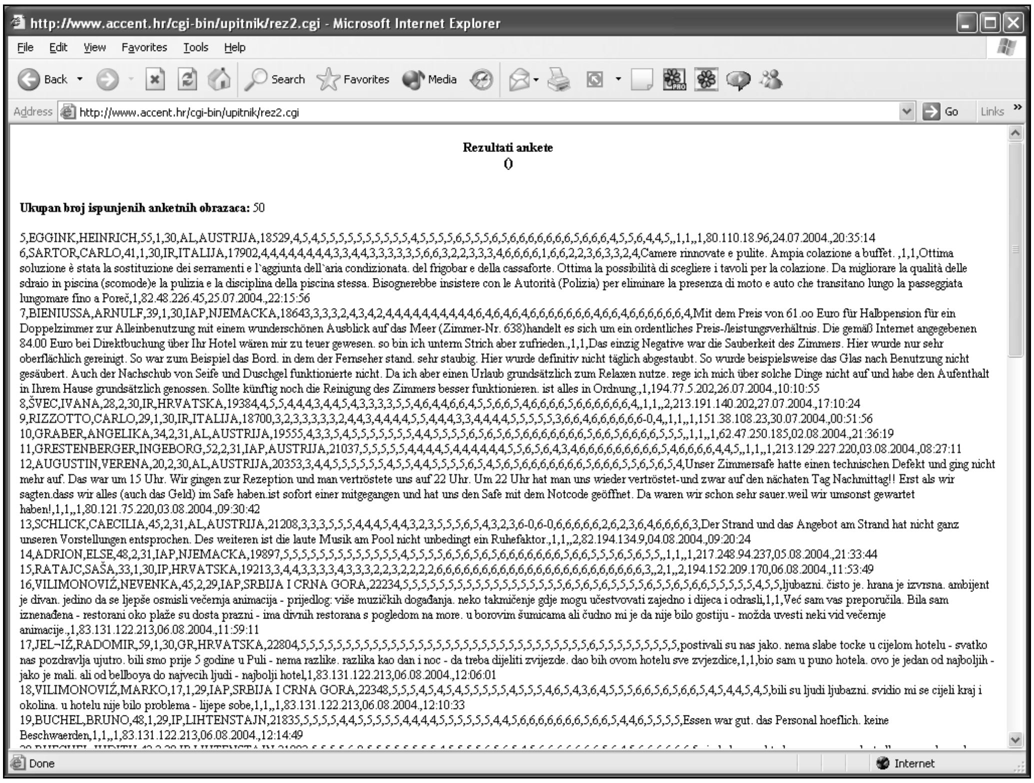
Slika 6.10: Prikaz ukupnih rezultata ankete u grafičkom obliku

Slika 6.11 prikazuje primjer djelomičnih rezultata ankete, koji se odnose na dobnu skupinu ispitanika od 36 do 45 godina (interval je određen parametrom dob: 1:0-17 godina, 2:18-25 godina, 3:25-35 godina, 4:36-45 godina, 5:46-55 godina i 6:56 i više godina).



Slika 6.11: Primjer djelomičnih rezultata ankete, za dob ispitanika između 36 i 45 godina

Slika 6.12: prikazuje primjer tekstualnih rezultata ankete. U tom obliku rezultati se mogu kopirati u tablični kalkulator radi daljnje obrade.



Slika 6.12: Primjer tekstualnih rezultata ankete

6.3 Prednosti generativnog razvoja aplikacija

U toku razvoja projekta obrađenog u poglavlju 6, kao i kod drugih projekata u okviru kojih je primijenjen generativan razvoj aplikacija utvrđene su slijedeće prednosti generativnog razvoja aplikacija u odnosu na klasične metode:

- brz razvoj sustava. Dio traženih funkcionalnosti sustava može se realizirati upotrebom postojećih generatora, bez potrebe za modifikacijama. Također, dijelovi opisa aplikacija mogu se preuzeti od prijašnjih projekata.
- jednostavne modifikacije sustava. U slučaju da su potrebne modifikacije u sustavu, u većini slučajeva dovoljne su promjene u opisima pojedinih aplikacija, pa nije potrebno modificirati sve programske datoteke.
- ponovna iskoristivost elemenata generativnog sustava. Razvijeni generatori, metaskripte i opisi aplikacija mogu se iskoristiti u okviru novih projekata generativnog razvoja aplikacija.

7 TESTIRANJE PRIMJENE KONCEPTA GENERATIVNOG PROGRAMIRANJA TEMELJENOG NA JEZICIMA SKRIPATA

Osnovni ciljevi koncepta generativnog programiranja temeljenog na jezicima skripata su slijedeći:

- skraćenje razvojnog ciklusa aplikacija,
- optimizacija performansi izrađenih aplikacija i
- pojednostavljenje održavanja aplikacija.

U okviru ovog poglavlja testirani su različiti generatori i postignuta svojstva izrađenih aplikacija obzirom na postignutu razinu jezika, performanse izrađenih aplikacija i pojednostavljenje održavanja.

7.1 Skraćenje razvojnog ciklusa aplikacija

Dosadašnja istraživanja pokazuju, prema (Ousterhout, 1998., str. 2) da visoka razina programskog jezika doprinosi efikasnosti programera, budući da korištenjem takvih jezika moraju za ostvarenje odgovarajućih funkcionalnosti, napisati manje programskog koda u odnosu na jezike niže razine. U slučaju programiranja u sistemskim programskim jezicima, kao što su Pascal ili C, razlikujemo izvorni i izvršni programski kod u strojnom jeziku. Za odgovarajući zadatak, potrebno je, otprilike 3-6 puta više programskih redaka u strojnom jeziku, nego u višem programskom jeziku, prema (Ousterhout, 1998., str. 2).

U slučaju generativnog programiranja temeljenog na jezicima skripata razlikujemo opis aplikacije i generirani programski kod. U slučaju da je veličina opisa aplikacije bitno manja od veličine generiranog programskog koda, možemo zaključiti da je jezik opisa aplikacije više razine od ciljnog programskog jezika aplikacije.

7.1.1 Testovi razine jezika

Razina programskog jezika može se definirati na dva načina, prema (Jones, 1997., str 2):
- brojem programskih instrukcija jezika niže razine (obično strojnog jezika) potrebnih za realizaciju jedne instrukcije zadanog programskog jezika ili
- brojem instrukcija zadanog programskog jezika za realizaciju jedne funkcijske točke programa.

Za mjerenje razine opisa aplikacije postoje određeni problemi u primjeni navedenih mjera razine jezika:
- pojedini dijelovi generiranog programskog koda ne moraju biti u istom ciljnom jeziku (npr. generatori web aplikacija mogu generirati programski kod u Perl-u i formulare za unos podataka u HTML-u),
- generirani programski kod može se koristiti na različitim strojnim konfiguracijama računala,
- jezik opisa aplikacije definira pojedina svojstva (aspekte) aplikacije, koji ne moraju predstavljati instrukcije, odnosno mogu se odnositi na procese i podatke.

Zbog toga je u ovom radu, prema (Jones, 1999., str. 1) definirana alternativna mjera razine opisa aplikacije. Ovdje je to omjer veličine programskog koda u ciljnom programskom jeziku u odnosu na veličinu programskog koda opisa aplikacije/komponente. Takva mjera unificira sve instrukcije i podatke korištene u različitim programskim jezicima svodeći ih na broj znakova odgovarajućih tekstualnih datoteka koje sadrže programski kod.

7.1.1.1 Omjer veličine programskog koda opisa aplikacija i odgovarajućih generiranih aplikacija

Ustanovljeni su odnosi veličine programskog koda opisa aplikacije i generirane aplikacije za slijedeće web aplikacije:

- aplikacija za anketiranje gostiju hotela (obrađena u poglavlju 6),
- aplikacija za održavanje baze ispitnih rokova i prezentaciju ispitnih rokova (obrađena u poglavljima 5.1.1.2 i 5.1.2.2),
- anketni upitnik (obrađen u poglavlju 5.1.3.1),
- jednostavan web portal,
- aplikacija za online testiranje korištenjem Spitzbergovih upitnika, primijenjenih unutar istraživanja o komunikacijskoj kompetenciji (Bubaš, Radošević i Hutinski, 2003.) i
- web testovi za C++.

7.1.1.1.1 Aplikacija za anketiranje gostiju hotela

Za izradu aplikacije za anketiranje gostiju hotela korištena su tri generatora: generator aplikacija za daljinsko održavanje baza podataka, generator korisničkih web aplikacija i generator anketnih upitnika. Generirani programski kod je u jezicima Perl i HTML. Tablica 7.1 prikazuje veličine programskog koda opisa aplikacije i generirane aplikacije:

generator	**veličina opisa (kb)**	**broj generiranih datoteka**	**veličina generiranog programskog koda (kb)**	**omjer generirani kod/opis**
daljinsko održavanje baza podataka	1,30	80	84,3	64,85
korisničke web aplikacije	6,22	2	10,5	1,69
anketni upitnici	4,02	5	160,41	39,90

Tablica 7.1: Omjer veličine opisa i generirane aplikacije kod aplikacije za anketiranje gostiju hotela

Ukupna veličina opisa aplikacija za sva tri korištena generatora iznosi 11,54 kb, dok ukupna veličina generiranog programskog koda iznosi 255.21 kb, što daje omjer veličine generiranog programskog koda i veličine opisa aplikacija od 22,11.

7.1.1.1.2 Aplikacija za održavanje baze ispitnih rokova i prezentaciju ispitnih rokova

Aplikacija za održavanje baze ispitnih rokova i prezentaciju ispitnih rokova (obrađena u poglavljima 5.1.1.2 i 5.1.2.2) generirana je korištenjem generatora aplikacija za daljinsko održavanje baza podataka i generatora korisničkih web aplikacija. Tablica 7.2 prikazuje veličine programskog koda opisa aplikacija i generirane aplikacije:

generator	**veličina opisa (kb)**	**broj generiranih datoteka**	**veličina generiranog programskog koda (kb)**	**omjer generirani kod/opis**
daljinsko održavanje baza podataka	0,53	35	34,0	97,14
korisničke web aplikacije	1,16	1	3,63	3,13

Tablica 7.2: Omjer veličine opisa i generirane aplikacije kod aplikacije za održavanje baze ispitnih rokova i prezentaciju ispitnih rokova

Ukupna veličina opisa aplikacija za oba korištena generatora iznosi 1,69 kb, dok ukupna veličina generiranog programskog koda iznosi 37,63 kb, što daje omjer veličine generiranog programskog koda i veličine opisa aplikacija od 22,26.

7.1.1.1.3 Anketni upitnik

Anketni upitnik (obrađen u poglavlju 5.1.1.3) generiran je korištenjem generatora anketnih web upitnika. Tablica 7.3 prikazuje veličine programskog koda opisa aplikacija i generirane aplikacije:

generator	**veličina opisa (kb)**	**broj generiranih datoteka**	**veličina generiranog programskog koda (kb)**	**omjer generirani kod/opis**
anketni upitnici	0,41	5	14,3	34,88

Tablica 7.3: Omjer veličine opisa i generirane aplikacije kod anketnog upitnika

7.1.1.1.4 Jednostavan web portal

Jednostavan web portal je aplikacija za održavanje i prezentaciju različitih informacija putem web sučelja, vidljiv je na (Radošević, 2003.). U izradi portala korištena su dva generatora, generator aplikacija za daljinsko održavanje baza podataka i generator korisničkih web aplikacija. Tablica 7.4 prikazuje veličine programskog koda opisa aplikacija i generirane aplikacije:

generator	**veličina opisa**	**broj**	**veličina**	**omjer**

	(kb)	generiranih datoteka	generiranog programskog koda (kb)	generirani kod/opis
daljinsko održavanje baza podataka	2,29	101	119,17	52,04
korisničke web aplikacije	14,3	1	18,8	1,31

Tablica 7.4: Omjer veličine opisa i generirane aplikacije kod jednostavnog web portala

Ukupna veličina opisa aplikacija za oba korištena generatora iznosi 16,59 kb, dok ukupna veličina generiranog programskog koda iznosi 137,97 kb, što daje omjer veličine generiranog programskog koda i veličine opisa aplikacija od 8,32.

7.1.1.1.5 Aplikacija za online testiranje korištenjem Spitzbergovih upitnika

Aplikacija za online testiranje korištenjem Spitzbergovih upitnika, primijenjenih unutar istraživanja o komunikacijskoj kompetenciji (Bubaš, Radošević i Hutinski, 2003.) izrađena je korištenjem dva generatora, generatora aplikacija za daljinsko održavanje baza podataka i generatora korisničkih web aplikacija. Tablica 7.5 prikazuje veličine programskog koda opisa aplikacija i generirane aplikacije:

generator	veličina opisa (kb)	broj generiranih datoteka	veličina generiranog programskog koda (kb)	omjer generirani kod/opis
daljinsko održavanje baza podataka	0,88	57	51,8	58,86
korisničke web aplikacije	5,56	1	8,72	1,57

Tablica 7.5: Omjer veličine opisa i generirane aplikacije kod jednostavnog web portala

Ukupna veličina opisa aplikacija za oba korištena generatora iznosi 6,44 kb, dok ukupna veličina generiranog programskog koda iznosi 60,52 kb, što daje omjer veličine generiranog programskog koda i veličine opisa aplikacija od 9,40.

7.1.1.1.6 Web testovi za C++

Generator web testova izrađen je modifikacijom generatora anketnih web upitnika. Korištenjem tog generatora izrađeno je nekoliko online testova za programski jezik C++ (Radošević, 2002.). Tablica 7.6 prikazuje veličine programskog koda opisa aplikacija i generirane aplikacije:

aplikacija	veličina opisa (kb)	broj generiranih	veličina generiranog	omjer generirani

		datoteka	**programskog koda (kb)**	**kod/opis**
teorijska pitanja	3,68	5	47,73	12,97
logičke (kontrolne) strukture	0,97	5	24,91	25,68
osnovni tipovi podataka i konverzije	0,89	5	24,54	27,57
funkcije	0,74	5	21,38	28,89
datoteke	1,50	5	26,42	17,61

Tablica 7.6: Omjer veličine opisa i generiranih aplikacija pomoću generatora web testova

Ukupna veličina opisa svih aplikacija za korišteni generator web testova iznosi 6,92 kb, dok ukupna veličina generiranog programskog koda svih aplikacija iznosi 144,98 kb, što daje prosječan omjer veličine generiranog programskog koda i veličine opisa aplikacija od 20,95.

7.1.1.1.7 Očekivani omjeri veličine opisa aplikacija i generiranog programskog koda

U okviru poglavlja 7.1.1.1 testirana su četiri generatora na ukupno 10 aplikacija. Po pojedinim generatorima aplikacija dobiveni su slijedeći rezultati, vidljivi u tablici 7.7:

generator	**broj aplikacija**	**ukupna veličina opisa aplikacija (kb)**	**ukupna veličina generiranog koda (kb)**	**omjer generirani kod/opis**
daljinsko održavanje baza podataka	4	5,0	289,27	57,85
korisničke web aplikacije	4	27,24	41,65	1,53
anketni upitnici	2	4,43	174,71	39,44
web testovi	5	6,92	144,98	20,95

Tablica 7.7: Omjeri veličine koda opisa aplikacija i generiranih aplikacija za testirane generatore

Prema tome, možemo vidjeti da prosječni omjeri veličine koda opisa aplikacija i generiranih aplikacija za testirane generatore variraju od 1,53 za generator korisničkih web aplikacija do 57,85 za generator aplikacija za daljinsko održavanja baza podataka.

Po aplikacijama dobivaju se slijedeći omjeri veličine koda opisa aplikacija i generiranih aplikacija (tablica 7.8):

aplikacija	**broj korištenih**	**ukupna veličina opisa**	**ukupna veličina**	**omjer generirani**

	generatora	aplikacija (kb)	generiranog koda (kb)	kod/opis
anketiranje gostiju hotela	3	11,54	255,21	22,11
održavanje baze ispitnih rokova i prikaz	2	1,69	37,63	22,26
anketni upitnik	1	0,41	14,3	34,88
jednostavan web portal	2	16,59	137,97	8,32
web upitnici prema Spitzbergu	2	6,44	60,52	9,40
web testovi za C++	1	6,92	144,98	20,95

Tablica 7.8: Omjeri veličine koda opisa aplikacija i generiranih aplikacija za testirane aplikacije

Možemo vidjeti da je raspon omjera veličine koda opisa aplikacija i generiranih aplikacija za testirane aplikacije manji (8,32 do 34,88) nego za pojedine generatore, što proizlazi iz korištenje dva ili više generatora za većinu aplikacija.

7.2 Optimizacija performansi izrađenih aplikacija

Testirane su generirane web aplikacije obzirom na brzinu izvršavanja i utjecaja složenosti aplikacije na brzinu izvršavanja. Kao mjera složenosti aplikacija uzeta je veličina opisa. Mjereno je vrijeme učitavanja web stranica pristupom web serveru preko Interneta. Mjerenja su izvršena u doba malog prometa na serveru, između 2 i 4h u noći. Aplikacije za testiranje dostupne su na adresi http://www.foi.hr/~darados/test/gotovo.

Korištena je slijedeća strojna oprema za testiranje:

- Internet poslužitelj: barok.foi.hr (DEC ALPHA, UNIX)

Lokalno računalo za testiranje:

- računalo Pentium Celeron 1 Ghz, 256 Mb Ram

Tablica 7.9 pokazuje korištene generatore aplikacija i veličine opisa:

generator	aplikacija	veličina opisa (kb)
daljinsko održavanje baza podataka	održavanje tablice s 3 polja	0,17
daljinsko održavanje baza podataka	održavanje tablice s 7 polja	0,25
daljinsko održavanje baza podataka	održavanje tablice s 12 polja	0,35
korisničke web aplikacije	prikaz stranice s 3 oznake	0,65

korisničke web aplikacije	prikaz stranice s 7 oznaka	1,23
korisničke web aplikacije	prikaz stranice s 12 oznaka	1,96
web upitnici	upitnik s 3 pitanja	0,17
web upitnici	upitnik s 7 pitanja	0,33
web upitnici	upitnik s 12 pitanja	0,54

Tablica 7.9: Korišteni generatori aplikacija i veličine opisa

Veza na Internet ostvarena je preko sustava kablovske televizije uz brzinu prijenosa podataka od 384/64 kbit/s (384 kbit/s od servera prema lokalnom računalu, a 64 kbit/s od lokalnog računala prema serveru).

7.2.1 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora aplikacija za daljinsko održavanje baza podataka

Mjerena su vremena učitavanja web stranica za različiti broj polja po tablici u bazi podataka i za različite operacije (ispis sadržaja tablice, prikaz obrasca za unos podataka, prikaz obrasca za ispravak podataka, te ekrana s ispisom podataka radi provjere kod brisanja). U svaku tablicu upisano je po 5 slogova. Izgled opisa aplikacije je slijedeći:

konekcija:$DBH=DBI->connect ("dbi:Pg:dbname=darados;host=alf.cat.foi.hr","darados","");

naslov:Prva
tablica:prva
polje_broj:polje1
polje_tekst:polje2
polje_broj:polje3

naslov:Druga
tablica:druga
polje_broj:polje1
polje_tekst:polje2
polje_broj:polje3
polje_tekst:polje4
polje_broj:polje5
polje_tekst:polje6
polje_broj:polje7

naslov:Treca
tablica:treca
polje_broj:polje1
polje_tekst:polje2
polje_broj:polje3
polje_tekst:polje4
polje_broj:polje5
polje_tekst:polje6
polje_broj:polje7
polje_tekst:polje8
polje_broj:polje9
polje_tekst:polje10
polje_broj:polje11

polje_tekst:polje12

naslov:predlosci
tablica:test_predlosci
polje_tekst:naziv_predloska
polje_memo:predlozak

Vremena učitavanja za tri generirane aplikacije prikazana su u tablici 7.10:

aplikacija	operacija	učitani sadržaj(kb)	vrijeme 1 (s)	vrijeme 2 (s)	vrijeme 3 (s)	prosjek (s)
održavanje tablice s 3 polja	ispis sadržaja tablice	0,74	3,37	2,91	2,67	2,98
održavanje tablice s 3 polja	prikaz obrasca za unos podataka	0,83	2,51	2,61	2,56	2,56
održavanje tablice s 3 polja	prikaz obrasca za ispravak podataka	0,93	2,72	3,21	2,82	2,92
održavanje tablice s 3 polja	prikaz pod. radi provjere kod brisanja	0,29	5,31	4,56	2,61	4,16
održavanje tablice s 7 polja	ispis sadržaja tablice	0,84	2,62	4,42	4,91	3,98
održavanje tablice s 7 polja	prikaz obrasca za unos podataka	1,08	3,36	2,67	3,83	3,29
održavanje tablice s 7 polja	prikaz obrasca za ispravak podataka	1,24	2,77	2,72	5,47	3,65
održavanje tablice s 7 polja	prikaz pod. radi provjere kod brisanja	0,30	2,67	2,67	6,18	3,84
održavanje tablice s 12 polja	ispis sadržaja tablice	0,98	5,18	2,66	4,42	4,09
održavanje tablice s 12 polja	prikaz obrasca za unos podataka	1,39	4,47	2,62	2,57	3,22
održavanje tablice s 12 polja	prikaz obrasca za ispravak podataka	1,62	2,71	2,84	2,76	2,77
održavanje tablice s 12 polja	prikaz pod. radi provjere kod brisanja	0,32	2,67	2,69	2,69	2,68

Tablica 7.10: Vremena učitavanja za tri generirane aplikacije za daljinsko održavanje baze podataka

U tablici 7.11 stavljena su prosječna vremena učitavanja stranica u odnos s veličinom učitanog sadržaja i s veličinom opisa aplikacije:

aplikacija	operacija	učitani sadržaj(kb)	veličina opisa (kb)	vrijeme učitavanja (s)	učitani sadržaj/vrijeme učitavanja (kb/s)	vel. opisa/vrijeme učitavanja (kb/s)
održavanje tablice s 3 polja	ispis sadržaja tablice	0,74	0,17	2,98	0,25	0,06
održavanje tablice s 3 polja	prikaz obrasca za unos podataka	0,83	0,17	2,56	0,32	0,07
održavanje tablice s 3 polja	prikaz obrasca za ispravak podataka	0,93	0,17	2,92	0,32	0,06
održavanje tablice s 3 polja	prikaz pod. radi provjere kod brisanja	0,29	0,17	4,16	0,07	0,04
održavanje tablice s 7 polja	ispis sadržaja tablice	0,84	0,25	3,98	0,21	0,06
održavanje tablice s 7 polja	prikaz obrasca za unos podataka	1,08	0,25	3,29	0,33	0,08
održavanje tablice s 7 polja	prikaz obrasca za ispravak podataka	1,24	0,25	3,65	0,34	0,07
održavanje tablice s 7 polja	prikaz pod. radi provjere kod brisanja	0,3	0,25	3,84	0,08	0,07
održavanje tablice s 12 polja	ispis sadržaja tablice	0,98	0,35	4,09	0,24	0,09
održavanje tablice s 12 polja	prikaz obrasca za unos podataka	1,39	0,35	3,22	0,43	0,11

održavanje tablice s 12 polja	prikaz obrasca za ispravak podataka	1,62	0,35	2,77	0,58	0,13
održavanje tablice s 12 polja	prikaz pod. radi provjere kod brisanja	0,32	0,35	2,68	0,12	0,13

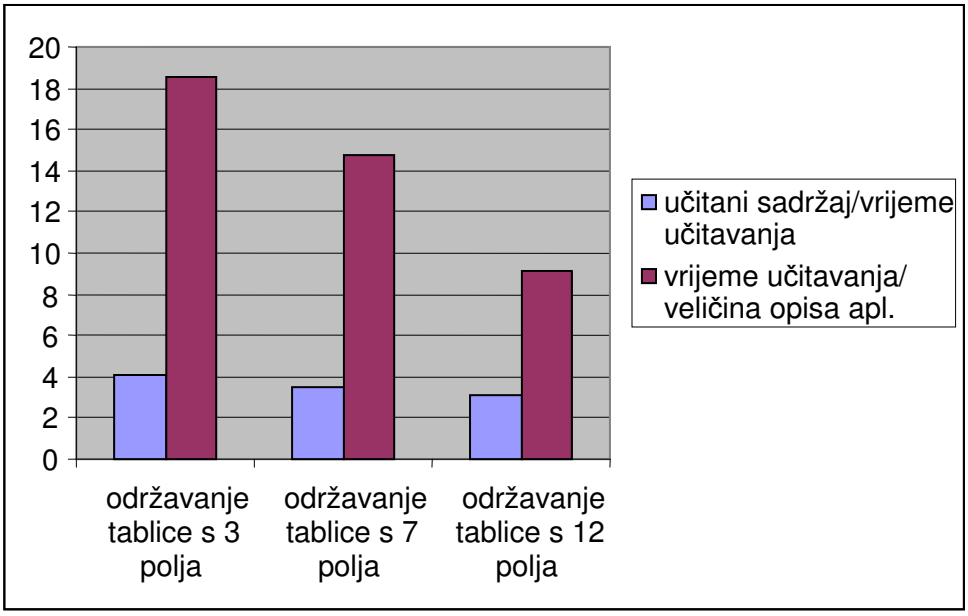
Tablica 7.11: Odnos prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije kod održavanja tablica s različitim brojem polja

Odnosno, prosječne vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije možemo vidjeti u tablici 7.12:

aplikacija	**prosjek učitanog sadržaja(kb)**	**veličina opisa (kb)**	**prosjek vremena učitavanja (s)**	**učitani sadržaj/vrijeme učitavanja (kb/s)**	**vrijeme učitavanja (kb/s)/vel. Opisa (kb)**
održavanje tablice s 3 polja	0,7	0,17	3,16	4,12	18,59
održavanje tablice s 7 polja	0,87	0,25	3,69	3,48	14,76
održavanje tablice s 12 polja	1,08	0,35	3,19	3,09	9,11

Tablica 7.12: Prosječne vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije

Vrijednosti iz tablice 7.12 možemo predočiti i grafički (grafikon 7.1):



Grafikon 7.1: Prosječne vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije

Možemo vidjeti da je prosjek vremena učitavanja približno jednak za sve tri aplikacije, odnosno, ne uočava se porast s povećanjem opisa aplikacije. Razlike u veličini učitanog sadržaja su premale da bi bitno utjecale na vrijeme učitavanja.

7.2.2 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora korisničkih web aplikacija

Mjerena su vremena učitavanja web stranica s različitim brojem oznaka koje se zamjenjuju sadržajem. Sadržaj web stranice definiran je sadržajem tri tablice korištene u poglavlju 7.2.1. U svaku tablicu upisano je po 5 slogova. Izgled opisa aplikacije je slijedeći:

```
konekcija:$DBH=DBI->connect ("dbi:Pg:dbname=darados;host=alf.cat.foi.hr","darados","");

globalna:vrsta,1

predlozak_stranice:select predlozak from test_predlosci where r_broj=$vrsta
++++++++++++++++++++++++++++++++++++++++
#polje1a#
sql:select polje1 from prva
b_prikaza:10000
polje:polje1
i:$zamjene="$zamjene,$polje1"
----------------------------------------
#polje2a#
sql:select polje2 from prva
b_prikaza:10000
polje:polje2
i:$zamjene="$zamjene,$polje2"
----------------------------------------
#polje3a#
sql:select polje3 from prva
```

```
b_prikaza:10000
polje:polje3
i:$zamjene="$zamjene,$polje3"
----------------------------------------
#polje1b#
sql:select polje1 from druga
b_prikaza:10000
polje:polje1
i:$zamjene="$zamjene,$polje1"
----------------------------------------
#polje2b#
sql:select polje2 from druga
b_prikaza:10000
polje:polje2
i:$zamjene="$zamjene,$polje2"
----------------------------------------
#polje3b#
sql:select polje3 from druga
b_prikaza:10000
polje:polje3
i:$zamjene="$zamjene,$polje3"
----------------------------------------
#polje4b#
sql:select polje4 from druga
b_prikaza:10000
polje:polje4
i:$zamjene="$zamjene,$polje4"
----------------------------------------
#polje5b#
sql:select polje5 from druga
b_prikaza:10000
polje:polje5
i:$zamjene="$zamjene,$polje5"
----------------------------------------
#polje6b#
sql:select polje6 from druga
b_prikaza:10000
polje:polje6
i:$zamjene="$zamjene,$polje6"
----------------------------------------
#polje7b#
sql:select polje7 from druga
b_prikaza:10000
polje:polje7
i:$zamjene="$zamjene,$polje7"
----------------------------------------
#polje1c#
sql:select polje1 from treca
b_prikaza:10000
polje:polje1
i:$zamjene="$zamjene,$polje1"
----------------------------------------
#polje2c#
sql:select polje2 from treca
b_prikaza:10000
polje:polje2
i:$zamjene="$zamjene,$polje2"
----------------------------------------
#polje3c#
sql:select polje3 from treca
```

```
b_prikaza:10000
polje:polje3
i:$zamjene="$zamjene,$polje3"
----------------------------------------
#polje4c#
sql:select polje4 from treca
b_prikaza:10000
polje:polje4
i:$zamjene="$zamjene,$polje4"
----------------------------------------
#polje5c#
sql:select polje5 from treca
b_prikaza:10000
polje:polje5
i:$zamjene="$zamjene,$polje5"
----------------------------------------
#polje6c#
sql:select polje6 from treca
b_prikaza:10000
polje:polje6
i:$zamjene="$zamjene,$polje6"
----------------------------------------
#polje7c#
sql:select polje7 from treca
b_prikaza:10000
polje:polje7
i:$zamjene="$zamjene,$polje7"
----------------------------------------
#polje8c#
sql:select polje8 from treca
b_prikaza:10000
polje:polje8
i:$zamjene="$zamjene,$polje8"
----------------------------------------
#polje9c#
sql:select polje9 from treca
b_prikaza:10000
polje:polje9
i:$zamjene="$zamjene,$polje9"
----------------------------------------
#polje10c#
sql:select polje10 from treca
b_prikaza:10000
polje:polje10
i:$zamjene="$zamjene,$polje10"
----------------------------------------
#polje1c#
sql:select polje11 from treca
b_prikaza:10000
polje:polje11
i:$zamjene="$zamjene,$polje11"
----------------------------------------
#polje1c#
sql:select polje12 from treca
b_prikaza:10000
polje:polje12
i:$zamjene="$zamjene,$polje12"
----------------------------------------
```

Vremena učitavanja za tri generirane aplikacije prikazana su u tablici 7.13:

aplikacija	učitani sadržaj(kb)	vrijeme 1 (s)	vrijeme 2 (s)	vrijeme 3 (s)	prosjek (s)
prikaz stranice s 3 oznake	0,84	4,86	3,16	2,73	3,58
prikaz stranice sa 7 oznaka	0,94	2,86	5,73	2,69	3,76
prikaz stranice s 12 oznaka	1,20	2,72	2,99	5,26	3,66

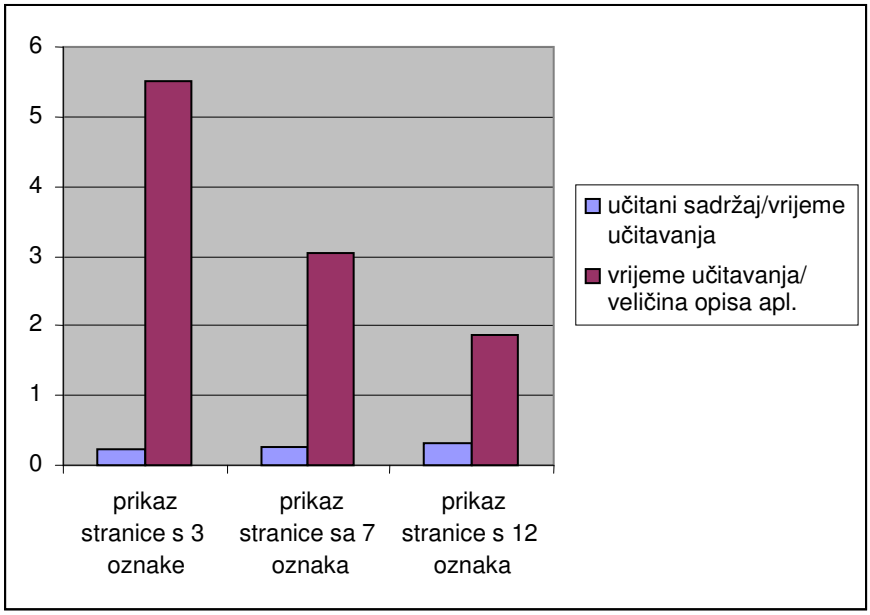
Tablica 7.13: Vremena učitavanja za tri generirane korisničke web aplikacije

Vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije možemo vidjeti u tablici 7.14:

aplikacija	učitani sadržaj (kb)	veličina opisa (kb)	prosjek vremena učitavanja (s)	učitani sadržaj/vrijeme učitavanja (kb/s)	vrijeme učitavanja (kb/s)/vel. Opisa
prikaz stranice s 3 oznake	0,84	0,65	3,58	0,23	5,51
prikaz stranice sa 7 oznaka	0,94	1,23	3,76	0,25	3,06
prikaz stranice s 12 oznaka	1,2	1,96	3,66	0,33	1,87

Tablica 7.14: Vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije

Vrijednosti iz tablice 7.14 možemo prikazati i grafički:



Grafikon 7.2: Vrijednosti omjera prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije za pojedine aplikacije

Možemo vidjeti da je, kao i u prethodnom primjeru iz poglavlja 7.2.1, prosjek vremena učitavanja približno jednak za sve tri aplikacije, odnosno, ne uočava se porast s povećanjem opisa aplikacije. Razlike u veličini učitanog sadržaja i ovdje su premale da bi bitno utjecale na vrijeme učitavanja.

7.2.3 Vrijeme učitavanja stranica kod aplikacija izrađenih pomoću generatora web upitnika

Mjerena su vremena učitavanja web stranica za tri upitnika, koji sadrže po 3, 7 odnosno 12 pitanja, i to kod tri različite operacije (unos odgovora u bazu podataka, prikaz grafičkih rezultata upitnika i prikaz rezultata upitnika u tekstualnom obliku). Izgled opisa upitnika za testiranje dat je u tablici 7.15:

upitnik s 3 pitanja	upitnik s 7 pitanja	upitnik s 12 pitanja
tablica:test_anketa1 *naslov:Test anketa 1* *pitanje_J:1. pitanje* *a* *b* *c* *pitanje_J:2. pitanje* *a* *b* *c* *pitanje_J:3. pitanje* *a* *b* *c* *polja:* *pitanje1* *pitanje2*	*tablica:test_anketa2* *naslov:Test anketa 2* *pitanje_J:1. pitanje* *a* *b* *c* *pitanje_J:2. pitanje* *a* *b* *c* *pitanje_J:3. pitanje* *a* *b* *c* *pitanje_J:4. pitanje* *a* *b*	*tablica:test_anketa3* *naslov:Test anketa 3* *pitanje_J:1. pitanje* *a* *b* *c* *pitanje_J:2. pitanje* *a* *b* *c* *pitanje_J:3. pitanje* *a* *b* *c* *pitanje_J:4. pitanje* *a* *b*

pitanje3	*c* *pitanje_J:5. pitanje* *a* *b* *c* *pitanje_J:6. pitanje* *a* *b* *c* *pitanje_J:7. pitanje* *a* *b* *c* *polja:* *pitanje1* *pitanje2* *pitanje3* *pitanje4* *pitanje5* *pitanje6* *pitanje7*	*c* *pitanje_J:5. pitanje* *a* *b* *c* *pitanje_J:6. pitanje* *a* *b* *c* *pitanje_J:7. pitanje* *a* *b* *c* *pitanje_J:8. pitanje* *a* *b* *c* *pitanje_J:9. pitanje* *a* *b* *c* *pitanje_J:10. pitanje* *a* *b* *c* *pitanje_J:11. pitanje* *a* *b* *c* *pitanje_J:12. pitanje* *a* *b* *c* *polja:* *pitanje1* *pitanje2* *pitanje3* *pitanje4* *pitanje5* *pitanje6* *pitanje7* *pitanje8* *pitanje9* *pitanje10* *pitanje11* *pitanje12*

Tablica 7.15: Izgled opisa upitnika za testiranje

Vremena učitavanja za tri generirana web upitnika prikazana su u tablici 7.16:

upitnik	**operacija**	**učitani sadržaj(kb)**	**vrijeme 1 (s)**	**vrijeme 2 (s)**	**vrijeme 3 (s)**	**prosjek (s)**
upitnik s 3 pitanja	unos odgovora u bazu pod.	0,19	2,96	3,25	3,20	3,14
upitnik s 3 pitanja	grafički prikaz	1,85	3,17	2,72	2,68	2,86

	rezultata					
upitnik s 3 pitanja	tekstualni prikaz rezultata	0,62	3,07	2,97	2,65	2,90
upitnik sa 7 pitanja	unos odgovora u bazu pod.	0,19	2,11	4,18	3,27	3.19
upitnik sa 7 pitanja	grafički prikaz rezultata	4,34	4,11	3,27	3,23	3,54
upitnik sa 7 pitanja	tekstualni prikaz rezultata	0,50	2,81	2,67	2,71	2,73
upitnik s 12 pitanja	unos odgovora u bazu pod.	0,19	3,37	2,77	3,45	3,20
upitnik s 12 pitanja	grafički prikaz rezultata	2,06	5,79	6,63	2,76	5,06
upitnik s 12 pitanja	tekstualni prikaz rezultata	0,77	2,71	2,77	2,81	2,76

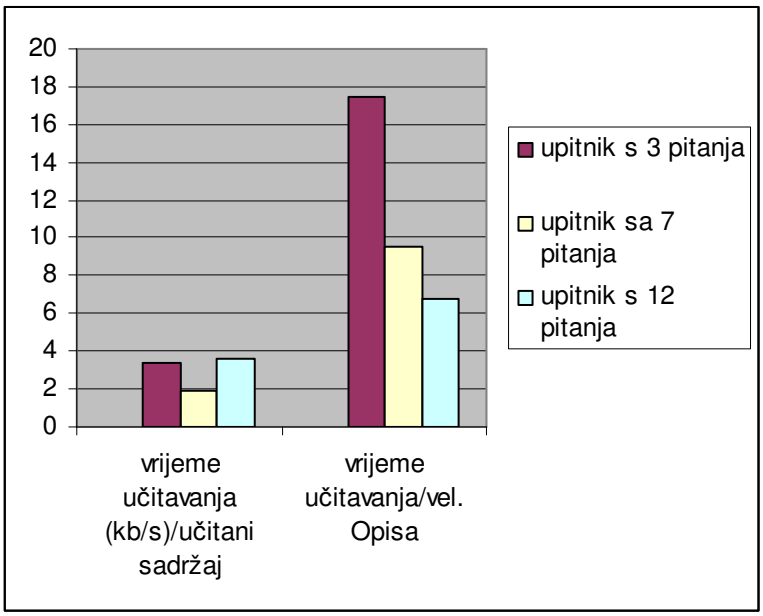
Tablica 7.16: Vremena učitavanja za tri generirana web upitnika

U tablici 7.17 stavljena su prosječna vremena učitavanja stranica u odnos s veličinom učitanog sadržaja i s veličinom opisa aplikacije:

upitnik	**prosjek učitanog sadržaja (kb)**	**veličina opisa (kb)**	**prosječno vrijeme učitavanja (s)**	**vrijeme učitavanja (kb/s)/učitani sadržaj**	**vrijeme učitavanja/vel. Opisa**
upitnik s 3 pitanja	0,89	0,17	2,97	3,34	17,47
upitnik sa 7 pitanja	1,68	0,33	3,15	1,88	9,55
upitnik s 12 pitanja	1,01	0,54	3,67	3,63	6,8

Tablica 7.17: Odnos prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije kod web upitnika s različitim brojem pitanja

Vrijednosti iz tablice 7.17 mogu se prikazati i grafički:



Grafikon 7.3: Odnos prosječnog vremena učitavanja stranica s veličinom učitanog sadržaja i veličinom opisa aplikacije kod web upitnika s različitim brojem pitanja

Za razliku od prethodna dva primjera, možemo vidjeti da, za razliku od prethodna dva primjera (poglavlja 7.2.1 i 7.2.2), prosjek vremena učitavanja umjereno raste s povećanjem opisa aplikacije. Razlike u veličini učitanog sadržaja i ovdje su premale da bi bitno utjecale na vrijeme učitavanja.

Međutim, kao i u prethodna dva primjera, odnos veličine opisa aplikacije i vremena učitavanja raste s povećanjem opisa aplikacije, pa možemo zaključiti da s povećanjem veličine opisa dobivamo relativno bolje performanse aplikacija obzirom na vrijeme učitavanja stranica.

7.3 Pojednostavljenje održavanja aplikacija

Održavanje je sastavni dio generativnog razvoja aplikacija, kao što možemo vidjeti na slici 5.1. Pojedine modifikacije nad aplikacijama mogu se izvršiti na tri razine:

- razina opisa aplikacije
- razina metaskripata
- razina generatora

U okviru ovog poglavlja uspoređen je broj potrebnih promjena za pojedine modifikacije aplikacija, ako se one izvrše na pojedinoj od navedene tri razine u odnosu na broj pojava tih promjena u generiranom programskom kodu. Testirane su aplikacije generirane pomoću ista tri generatora koji su korišteni za testiranja u poglavlju 7.2.

7.3.1 Pojednostavljenje održavanja aplikacija na razini modifikacija opisa aplikacije

Testirane su aplikacije izrađene pomoću različitih generatora na način da su u opis aplikacije dodane nove oznake, što je kod različitih generatora imalo različite učinke na generirani programski kod:

- uvedeno je novo polje u pojedinu tablicu - kod generatora aplikacija za daljinsko održavanje baze podataka,
- uvedena je nova zamjenska oznaka - kod generatora korisničkih web aplikacija i
- uvedeno je novo pitanje - kod generatora web upitnika

Tablica 7.18 pokazuje korištene generatore aplikacija, izvršene modifikacije i broj promjena u generiranom programskom kodu:

generator	**modifikacija**	**broj pojava u generiranom kodu**
daljinsko održavanje baza podataka	dodavanje polja	26
korisničke web aplikacije	dodavanje zamjenske oznake	2
web upitnici	dodavanje pitanja	23

Tablica 7.18: Korišteni generatori aplikacija, izvršene modifikacije i broj promjena u generiranom programskom kodu

Vrijednosti broja pojava u generiranom kodu identične su za različite aplikacije generirane pomoću istog generatora. Ove vrijednosti možemo usporediti s vrijednostima u tablici 7.7. Generatori koji postižu veće prosječne omjere veličine koda opisa aplikacija i generiranih aplikacija imaju i veći broj pojava pojedinih promjena u generiranom programskom kodu.

7.3.2 Pojednostavljenje održavanja aplikacija na razini modifikacija predložaka programskog koda

U slučaju da se tražene modifikacije aplikacija ne mogu postići samo promjenama njihovog opisa, modifikacijama metaskripata moguće je utjecati na svojstva generatora, kao što su:

- uključivanje novih funkcionalnosti i svojstava
- promjena implementacije postojećih funkcionalnosti i svojstava

Pojedina promjena u određenoj metaskripti imat će toliko pojava u generiranoj aplikaciji koliko puta pojedini generator koristi tu metaskriptu u generiranju programskog koda (vidljivo iz dijagrama metaskripata). Međutim, izbjegavaju se direktne promjene u generiranom programskom kodu, pa se pojednostavljenje održavanja postiže tako da promjene u metaskriptama ostaju sačuvane za slijedeća generiranja aplikacije, uključujući i modifikacije u opisu aplikacije.

7.3.3 Pojednostavljenje održavanja aplikacija na razini modifikacija generatora

Modifikacije generatora moguće su na dva načina:

- kroz modifikacije funkcija za predobradu podataka iz opisa aplikacije i
- kroz modifikacije skriptnog modela generatora.

Modifikacije funkcija za predobradu podataka iz opisa aplikacije imaju toliko pojava u generiranoj aplikaciji koliko puta pojedini generator koristi tu funkciju u generiranju programskog koda, što je vidljivo iz dijagrama metaskripata.
Modifikacije skriptnog modela proizlaze iz potrebe da se opis aplikacija obogati mogućnošću zadavanja novih svojstava, te iz potrebe za promjenom funkcionalnosti generiranih aplikacija koje proizlaze iz tih novih svojstava.

I u slučaju modifikacija generatora, kao i kod modifikacije metaskripata, izbjegavaju se direktne promjene u generiranom programskom kodu, pa se pojednostavljenje održavanja postiže tako da promjene u metaskriptama ostaju sačuvane za slijedeća generiranja aplikacije, uključujući i modifikacije u opisu aplikacije, te metaskriptama.

8 ZAKLJUČAK

Istraživanja provedena u okviru ovog rada imala su za cilj definiranje koncepta generativnog programiranja temeljenog na jezicima skripata, umjesto na objektno-orijentiranim programskim jezicima, kako je zamišljeno od temeljnih autora na području generativnog programiranja (D. Batory, K. Czarnecky, U. Eisenecker i drugi). Takav koncept trebao bi iskoristiti određena svojstva jezika skripata koja se mogu iskoristiti za izgradnju generatora aplikacija, a koja se teško mogu postići korištenjem klasičnih, sistemskih programskih jezika. Utvrđeno je da se radi o slijedećim svojstvima jezika skripata:

- visoka razina jezika skripata, zajedno s niskim stupnjem tipizacije pojednostavljuju izgradnju i kasnije modifikacije generatora u okviru generativnog razvoja aplikacija,
- pojedini jezici skripata, kao što je Perl, znatno pojednostavljuju obradu znakovnih nizova što omogućuje jednostavniju implementaciju funkcija generiranja,
- visoka fleksibilnost jezika skripata omogućuje korištenje polja neograničene dužine i funkcija s neograničenim brojem parametara, što pojednostavljuje izradu generatora,
- mogućnost samoevaluacije (izvršavanja znakovnih nizova kao programskog koda) je svojstvo koje se teško može postići kod sistemskih programskih jezika koji razlikuju izvorni od izvršnog programskog koda. Upravo mogućnost samoevaluacije otvara mogućnost kasnijeg razvoja novih arhitektura generatora koji bi generirani programski kod smještali u radnu memoriju (npr. u polja neograničene dužine) gdje bi se mogao i izvršavati, budući da nema potrebe za prevođenjem.

U toku istraživanja u okviru ovog rada u svijetu su se pojavila dva projekta koji se također bave problematikom primjene jezika skripata u okviru generativnog programiranja:

- projekt razvoja jezika Open Promol, u Litvi i
- projekt Codeworker, u Francuskoj.

Oba projekta usmjerena su na razvoj novih programskih jezika, koji pripadaju jezicima skripata, specijaliziranih za izradu generatora aplikacija.
Za razliku od tih projekata u ovom radu naglasak je stavljen na razvoj grafičkog modela koji omogućuje modeliranje generatora, te na postupak generativnog razvoja aplikacija, a ne na izradu novog programskog jezika. U toku istraživanja pokazalo se da neki od jezika skripata, u prvom redu Perl, već sadrže odgovarajuća poželjna svojstva za izradu generatora aplikacija, odnosno, mogu se iskoristiti u okviru generativnog programiranja.

Zbog toga se **znanstveni doprinos** ovog rada sastoji u slijedećem:

- primjena jezika skripata u okviru generativnog programiranja je do sada malo istraženo područje, jer se većina dosadašnjih istraživanja (osim navedenih projekata Open Promol i Codeworker), te radovi glavnih istraživača na području generativnog programiranja (Don Batory, Ulrich Eisenecker, Krzysztof Czarnecky i drugi) odnose na primjenu principa generativnog programiranja na razvoj generatora i aplikacija u jezicima objektno-orijentiranog programiranja. Također, modeliranje na području jezika skripata je do sada malo istraženo područje. U okviru ovog rada definiran je koncept generativnog programiranja temeljen na jezicima skripata, te u okviru toga:

- definiran je skriptni model generatora aplikacija, kao grafički model generatora aplikacija prema predloženom konceptu generativnog programiranja, temeljenom na

jezicima skripata. Izrađeni grafički model uspoređen je s postojećim objektnim modelom koji se temelji na UML-u i
- definiran je generativni razvoj aplikacija, kao postupak razvoja programa i generatora aplikacija prema predloženom konceptu generativnog programiranja. Takvim postupkom omogućeno je skraćivanje razvojnog ciklusa i jednostavnije održavanje razvijenih aplikacija.

Utoliko se ovaj rad razlikuje od navedenih projekata vezanih uz primjenu jezika skripata u okviru generativnog programiranja (Open Promol i Codeworker).

U odnosu na **hipoteze**, navedene u poglavlju 1.2, utvrđeno je slijedeće:

1. Provedena istraživanja pokazala su da danas aktualni principi programiranja, generativno programiranje i jezici skripata, mogu evoluirati u novi koncept generativnog programiranja za jezike skripata. Razvijeni koncept sadrži naslijeđena svojstva navedenih principa programiranja, ali i neka nova svojstva i prednosti, čime su izbjegnute određene slabosti postojećeg koncepta generativnog programiranja, koji se temelji na objektno-orijentiranim programskim jezicima:

- izbjegava se problem složene sintakse i visoke razine tipizacije objektno-orijentiranih programskih jezika,
- izostavlja se faza prevođenja u razvoju generatora,
- pojednostavljene su promjene u kodu generatora, u okviru generativnog razvoja aplikacija, jer je programski kod u jezicima skripata više razine od koda u sistemskim programskim jezicima.

2. Generator je prema razvijenom konceptu generativnog programiranja potpuno odvojen od koda generirane aplikacije, odnosno, generirana aplikacija je potpuno neovisna o generatoru.

3. U razvijenom konceptu generativnog programiranja koriste se specifične mogućnosti jezika skripata, za izradu generatora aplikacija:

- visoka razina jezika skripata, zajedno s niskim stupnjem tipizacije omogućuje jednostavnije modifikacije generatora u okviru generativnog razvoja aplikacija,
- mogućnosti jezika skripata u obradi znakovnih nizova omogućuju jednostavniju implementaciju funkcija generiranja,
- visoka fleksibilnost jezika skripata koja uključuje korištenje polja neograničene dužine te funkcija s neograničenim brojem parametara također pojednostavljuje izradu generatora,
- mogućnost samoevaluacije (izvršavanja znakovnih nizova kao programskog koda).

Prema tome, možemo zaključiti da su hipoteze potvrđene.

U odnosu na **ciljeve**, navedene u poglavlju 1.1, postignuto je slijedeće:

1. Kreiran je novi pristup programiranju koji se temelji na generativnom programiranju u jezicima skripata, što rezultira skraćenjem razvojnog ciklusa aplikacija, optimizacijom performansi i pojednostavljenjem održavanja, što je bio glavni cilj izrade ovog rada.
U okviru razvijenog koncepta definirani su slijedeći elementi:

- arhitektura sustava za generativno programiranje,
- skriptni model generatora,

- generativni razvoj aplikacija, kao postupak u kojem se paralelno izrađuju aplikacije i razvijaju generatori,

Osim toga, razvijeno je nekoliko generatora prema skriptnom modelu i provedeno je testiranje razvijenih generatora obzirom na postignuta svojstva vezana uz:

- skraćivanje razvojnog ciklusa aplikacija,
- optimizaciju programskog koda generiranih aplikacija i
- pojednostavljenje održavanja aplikacije.

2. U odnosu na podciljeve, postignuto je slijedeće:

- na temelju istraživanja aktualnih tehnologija na području programiranja, u prvom redu jezika skripata, te postojećeg koncepta generativnog programiranja definiran je novi koncept generativnog programiranja, koji se temelji na oba područja,
- u okviru novog koncepta generativnog programiranja, koji se temelji na jezicima skripata definirana je arhitektura sustava za generiranje aplikacija i postupak generiranja, a istražene su i mogućnosti automatizacije izrade generatora aplikacija kroz parametrizaciju pojedinih dijelova i izradu jednostavnog metageneratora,
- primjena koncepta generativnog programiranja temeljenog na jezicima skripata testirana je na izradi nekoliko generatora aplikacija. Izrađeni generatori su testirani obzirom na postignuta svojstva, vezana uz skraćivanje razvojnog ciklusa aplikacija, optimizaciju programskog koda generiranih aplikacija i pojednostavljenje održavanja aplikacija,
- utvrđena je primjenjivost koncepta generativnog programiranja temeljenog na jezicima skripata na razvoj Web aplikacija u jezicima skripata.

Osim ovih ciljeva definiran je postupak generativnog razvoja aplikacija, kao iterativni postupak unutar kojeg se paralelno razvijaju aplikacije i generatori aplikacija, što rezultira skraćivanjem razvojnog ciklusa aplikacija, optimizacijom programskog koda generiranih aplikacija i pojednostavljenjem održavanja.

Od navedenih ciljeva, po pojedinim poglavljima rada ostvareno je slijedeće:

U uvodnom dijelu definirani su ciljevi rada, hipoteze, postignuti rezultati sa znanstvenim doprinosom i struktura rada.

Drugo poglavlje daje pregled aktualnih tehnologija i trendova na području programiranja, uključujući temeljne discipline ovog rada: jezike skripata i generativno programiranje. Ukazano je na temeljne razlike između generativnog programiranja kao relativno novog pristupa u programiranju u odnosu na tradicionalni, generički pristup, te na specifičnosti primjene jezika skripata.

Treće poglavlje daje pregled dostignuća dvaju dosadašnjih projekata primjene jezika skripata u okviru generativnog programiranja (Open promol i Codeworker). U okviru ta dva projekta razvijeni su specijalizirani jezici skripata namijenjeni za izradu generatora aplikacija.

Četvrto poglavlje je centralno u okviru ovog rada jer obrađuje razvijeni skriptni model generatora koji se temelji na dva grafička dijagrama:

- dijagramu parametara opisa aplikacija. Dijagram parametara opisa aplikacije je hijerarhijski dijagram koji definira zadana svojstva (aspekte) aplikacije i

- dijagramu metaskripata. Dijagram metaskripata definira distribuciju pojedinih specifičnih svojstava, zadanih dijagramom parametara opisa aplikacije, unutar aplikacije, te njihovo povezivanje s predlošcima programskog koda (metaskriptama).

U okviru tog poglavlja obrađeni su osnovni pojmovi novog koncepta generativnog programiranja, te je razvijen primjer jednostavnog generatora aplikacija u jeziku C++. Data je gramatika skriptnog modela, te je razvijeni skriptni model stavljen u odnos s postojećim objektnim modelom, koji se temelji na UML-u.

Peto poglavlje obrađuje generativni razvoj aplikacija kao iterativni proces u okviru kojeg se paralelno razvijaju aplikacije i generatori aplikacija. Ukazano je na prednosti takvog razvoja aplikacija, prvenstveno zbog ponovne iskoristivosti generatora i pojednostavljenja održavanja aplikacija.

Šesto poglavlje obrađuje jedan projekt primjene generativnog razvoja aplikacija. U okviru tog projekta razvijena je web aplikacija za ispitivanje zadovoljstva gostiju jednog hotela.

Sedmo poglavlje odnosi se na testiranje primjene koncepta generativnog programiranja temeljenog na jezicima skripata. Testovi se odnose na skraćivanje razvojnog ciklusa aplikacija, optimizaciju performansi generiranih aplikacija i pojednostavljenje održavanja aplikacije.

U prilogu je dat pojmovnik generativnog programiranja prema konceptu temeljenom na jezicima skripata.

Prema tome, možemo zaključiti da je definiranjem koncepta generativnog programiranja temeljenog na jezicima skripata, te pripadajućeg generativnog razvoja aplikacija, postignut očekivani znanstveni doprinos ovog rada, što je bio osnovni cilj. Također, polazne hipoteze pokazale su se točnim.

LITERATURA

[1] Aaby, A.A.:" Introduction to Programming Languages", Walla Walla College, Computer Science Department,1996., http://cs.wwc.edu/~aabyan/221_2/PLBOOK/, preuzeto 31.07.2002.
[2] Ambler S.W.: "The Diagrams of UML 2.0", http://www.agilemodeling.com/essays/umlDiagrams.htm, Agile Modeling, 2004., preuzeto 23.06.2004.
[3] Attardy, G., Cisternino, A.: " Reflection support by means of template metaprogramming ", Lecture Notes in Computer Science, Vol. 2186, 2001., http://www.di.unipi.it/~attardi/Paper/GCSE01.pdf, preuzeto 13.06.2004.
[4] Back, G.:"DataScript - A specification and Scripting Language for Binary Data", Proceedings of the Generative Programming and Component Engineering conference (GPCE 2002), Pittsburg, PA, USA, 2002.
[5] Barca, M.: "Aspect Oriented Programming, With an Overview of AspectJ", CIS Department of Ohio State University, 2003., http://www.cse.ohio-state.edu/~barca/PRESENTATIONS/AOP.ppt, preuzeto 24.06.2004.
[6] Batory, D., Johnson C., MacDonald B, Heeder, D.V.: "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study", ACM Transactions on Software Engineering and Methodology, Vol. 111,, No. 2, april 2002, ftp://ftp.cs.utexas.edu/pub/predator/fsatsRevised.pdf, preuzeto 3.10.2002.
[7] Bezroukov, N.:"A Slightly Skeptical View on Scripting Languages", http://www.softpanorama.org/Scripting/index.shtml, preuzeto 15.06.2004.
[8] Bubaš G., Radošević D., Hutinski Ž.: Assessment of Computer Mediated Communication Competence: Theory and Application in an Online Environment", Journal of Information and Organizational Sciences (JIOS),Vol. 27, NO. 2, Fakultet organizacije i informatike, Varaždin, 2003.
[9] Budd T. A. : "Object-Oriented programming", poglavlje 1 iz knjige Salus P.:"HPL: Vol. I Object-Oriented Programming Languages", Macmillan Computer publishing, 1998.
[10] Chroboczek J. : “Review of existing Languages”, http://cecc.gsnu.ac.kr/~astruct/unix/langua~1.htm, preuzeto 10.10.1998.
[11] Cilia, M., Haupt, M., Mezini, M., Buchmann, A.:"The Convergence of AOP and Active Databases: Towards Reactive Middleware", Proceedings of the Generative Programming and Component Engineering conference (GPCE 2003), Erfurt, Germany, 2003.
[12] Cordy, J.R., Shukla M.: "Practical Metaprogramming", Queenn's University, Kingston, Canada, 1992., http://www.cs.queensu.ca/TechReports/Reports/1992-342.pdf, preuzeto 29.07.2002.
[13] Crowe, W.L. : "Comparing C++ and Java: A Java switchboard implementation", C/C++ Users Journal, San Francisko, January 1999.
[14] Czarnecki K., Eisenecker U., Glück R., Vandevoorde D., Veldhuizen T.:"Generative Programming and Active Libraries", http://osl.iu.edu/~tveldhui/papers/dagstuhl1998/, preuzeto 30.07.2002.
[15] Czarnecki K., Eisenecker, U.W.: "Components and Generative Programming",http://www-ia.tu-ilmenau.de/~czarn/esec99/esec99.pdf, ECOOP’2000 Workshop on “Aspects and Dimensions of Concerns” ,2000., preuzeto 3.10.2002.
[16] Czarnecki K.: "Brief Overview of Generative Programming", DaimlerChrysler Research and Technology Ulm, Germany, 2002., http://www.cwi.nl/events/2002/GP2002/slides/czarnecki-gp-intro.pdf, preuzeto 29.07.2002.

[17] Czarnecki, K.: "Generative Programming and GMCL", Technische Universität Ilmenau,Fakultät für Informatik und Automatisierung, 1999., http://www-ia.tu-ilmenau.de/ ~czarn/gmcl/, preuzeto 29.07.2002.
[18] Czarnecki, K., Eisenecker, U. W.:"Generative programming: methods, tools and applications", Addison-Wesley, 2000.
[19] Czarnecki, K.:"Generative Core Concepts - Generative Domain Model", Program-Transformation.Org, 2002., http://www.program-transformation.org/Transform/ GenerativeCoreConcepts, preuzeto 25.06.2004.
[20] Delta Software: " Background Information: Generative Programming – From Theory to Practice", Delta Software Technology GmbH, 2004., http://www.d-s-t-g.com/neu/pages/ pageseng/service/doc/delta-aktuell/gp_hintergrnd.htm, preuzeto 14.06.2004.
[21] Devis, R.: "The Object-Oriented Page", http://www.well.com/user/ritchie/oo.html, zadnje ažuriranje 20.06.1997., preuzeto 28.06.2004.
[22] Dictionary.com : "The American Heritage® Dictionary of the English Language, Fourth Edition", Houghton Mifflin Company, www.dictionary.com, preuzeto 14.06.2004.
[23] Dumičić K, Sajko M, Radošević D.: "Designing a Web-Survey Questionnaire Using Automatic Process and a Script Language", Zbornik radova konferencije "IIS 2002", Fakultet organizacije i informatike, Varaždin, 2002.
[24] Eichelberger, H., Wolff, J.:"UML Description of the STL", First Workshop on C++ Template Programming, Erfurt, Germany, 2000., http://citeseer.ist.psu.edu/eichelberger00uml.html
[25] Eisenecker U.:"Generative Programming (GP) with C++", Proceedings of Modular Programming Languages (JMLC'97, Linz, Austria, March 1997.), Springer-Verlag, heidelberg 1997.
[26] Eisenecker, U. "Generative Programming: Beyond Generic Programming", Proc. Dagstuhl Seminar on Generic Programming, April 27--May 1, 1998, Schloß Dagstuhl,Wadern, Germany, 1998.
[27] Fedorov A., Francis B., Harrison R., Homer A., Murphy S., Smith R., Sussman D., Wood S.:"Professional Active Server Pages 2.0", Wrox Press, Birmingham, 1998.
[28] Flanagan, D:"Java in a Nutshell, Second Edition", O'Reilly, Sebastopol, CA, USA, 1997.
[29] Ford S., Wells D., Wells N.:"Web Programming Languages", Object Services and Consulting, Inc., 1997., http://www.objs.com/survey/lang.htm, preuzeto 1.08.2002.
[30] Gibbons, J., Jeuring, J.:"Generic programming", IFIP International Federation for Information Processing, siječanj 2003., http://www.neutrino.co.jp/abi_ifip/1-4020-7374-7.PDF, preuzeto 14.06.2004.
[31] Givargis, T., Vahid, F.: "Parametrized System Design", Proceedings of 8th International Workshop on Hardware/Software Codesign, (CODES'2000), 2000., str. 98-102
[32] Gomaa, H.: "A Domain Analysis and Modeling Method: Application to NASA's Payload Operations Control Center Domain", Department of Information and Software Systems Engineering, George Mason University, Fairfax, Virginia, 1997., http://sunset.usc.edu/gsaw/gsaw97/Gomaa.pdf, preuzeto 19.06.2004.
[33] Graham, I.S.:"The HTML Sourcebook", John Wiley & Sons, New York, 1995.
[34] Gray J., Lin Y., Zhang J.: "Levels of Independence in Aspect-Oriented Modeling", http://www.gray-area.org/Pubs/middleware-2003.pdf, preuzeto 15.06.2004.
[35] Griss, M.L.:"Implementing Product-Line Features with Component Reuse", Proceedings of the 6th International Conference on Software Reuse, Vienna, Austria, lipanj 2000., http://www.hpl.hp.com/reuse/papers/icsr2000-griss.pdf, preuzeto 14.06.2004.
[36] Guerraoui R. : "Strategic directions in object-oriented programming", ACM Computing Surveys, Baltimore, december 1996.

[37] HankinHanne C., Riis N., Palsberg J. : "Strategic directions in research on programming languages", ACM Computing Surveys, Baltimore, december 1996.
[38] Hejlsberg A., Wiltamuth S. : "C# Language Reference", Microsoft Corporation, 2000.
[39] Hermann A.C., Gaspary L.P., Almeida J.B. : “Design and Execution of Adaptive Multimedia Applications in the Internet”, ACM Computing Surveys, april 1998.
[40] Hertz P. : "Programming in Lingo", http://collaboratory.nunet.net/phertz/portfoli/c79notes/lingo.htm, Northwestern University, 1999.
[41] Jones, C. : "Programming Languages Table", Software Productivity Research, Inc., 1997., http://www.spr.com/library/0langtbl.htm, preuzeto 05.10.1998.
[42] Jones, C. : " Software Metrics & Metrics Counsel", Software Productivity Research, Inc., 1999., http://www.spr.com/metrics/, preuzeto 31.07.2002.
[43] Joyner I.: "C++?? : A Critique of C++", Prentice Hall, 1996., http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3.html, preuzeto 31.07.2002.
[44] Kanavin, A.:"An overview of scripting languages", Sensi.org, 2002., http://www.sensi.org/~ak/impit/studies/report.pdf, preuzeto 25.06.2004.
[45] Kandé, M.M., Kienzle,J., Strohmeier, A.: "From AOP to UML - A Bottom-Up Approach", 1st International Conference on Aspect-Oriented Software Development, 2002., Enschede, The Netherlands, http://lglwww.epfl.ch/workshops/aosd-uml/Allsubs/kande.pdf, preuzeto 23.06.2004.
[46] Kinczales, G., Lamping, J., Mendhekar, A., Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. “Aspect-Oriented Programming”. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997. http://citeseer.nj.nec.com/kiczales97aspectoriented.html, preuzeto 21.01.2004.
[47] Kleinöder J., Golm M. : "Transparent and Adaptable Object Replication Using a Reflective Java", University of Erlangen-Nürnberg, 1996., http:// www4.informatik.uni-erlangen.de/TR/pdf/TR-I4-96-07.pdf, preuzeto 29.07.2002.
[48] Kliček B., Radošević D.:"Development of the Multimedial Scripting Language of Higher Level", zbornik radova konferencije "ITI 2001",Pula, Fakultet elektrotehnike i računarstva, Zagreb,2001.
[49] Kühl, D.:" STL and OO Don't Easily Mix", Proceedings of the GSCE, Workshop on C++ Template Programming, Erfurt, 2000., http://www.oonumerics.org/tmpw00/ kuehl.html, preuzeto 23.06.2004.
[50] Lee, K.W.K.:"An Introduction to Aspect-Oriented Programming", COMP610E: Course of Software Development of E-Business Applications (Spring 2002), Hong Kong University of Science and Technology, 2002.,
[51] Lemaire, C.: "CODEWORKER Parsing tool and Code generator - User’s guide & Reference manual", http://codeworker.free.fr/CodeWorker.pdf, CodeWorker.free.fr, 2003., preuzeto 5.04.2004.
[52] Levenez, E.:"Computer Languages History", http://www.levenez.com/lang/, preuzeto 28.06.2004.
[53] Lieberherr, K.J.:"General AOP: Join Point Model", Demeter, Center for Software Sciences, 2003., http:// www.ccs.neu.edu/research/demeter/ course/w03/lectures/ General%20AOP.ppt, preuzeto 24.06.2004.
[54] Malinowski A., Wilamowski B. M.:" Web-based C++ Compiler", Bradley University, Peoria, IL / University of Wyoming, Laramie, WY, 2000., http://nn.uidaho.edu/pap/2000/ASEE2000_Complilers.pdf, preuzeto 1.08.2002.
[55] McGrath S. : "The Tcl/Tk and Python Scripting Environments", Dr. Dobb's Journal, October 1998.
[56] Meijer E., Gough J.: "Technical Overview of the Common Language Runtime", preuzeto s http:// research.microsoft.com/~emeijer/Papers/CLR.pdf, preuzeto 29.07.2002.

[57] Milićev, D.: "Objektno orijentisano modelovanje na jeziku UML", Mikro knjiga, Beograd, 2001.
[58] Morin R., Brown V. : “Scripting languages”, Sunexpert, september 1998.
[59] Mosses, P.D.: "Compiler Generation", University of Aarhus, Centre for Basic Research in Computer Science, Aarhus, Danmark, 2002., http://www.brics.dk/~pdm/PAGT-02/slides/intro.pdf, preuzeto 31.07.2002.
[60] Motik B., Šribar J. : "Demistificirani C++", Element, Zagreb, 1997.
[61] *O'Reilly, T. : "*The Importance of Perl", O'Reilly & Associates, Inc., 1998., http://perl.oreilly.com/news/importance_0498, preuzeto 31.07.2002.
[62] Ousterhout J. K. : "Scripting : Higher Level programming for the 21st Century", IEEE computer magazine, march 1998.
[63] Plösch R.: "Design by Contract for Python", Ch. Doppler Laboratory for Software Engineering, Johannes Kepler University of Linz, 1997., http://www.swe.uni-linz.ac.at/publications/pdf/TR-SE-97.24.pdf, preuzeto 29.07.2002.
[64] Prasad, M.D.:"Typecasting As a New Join Point in AspectJ", Proceedings of the 17th European Conference on Object-Oriented Programming (ECOOP), Damstadt, Germany, 2003., http://quercusseg.unex.es/jclemente/phdoos03/papers/paper1.pdf, preuzeto 24.06.2004.
[65] Radošević D:"Multimedijski jezik skripata više razine", magistarski rad, Fakultet organizacije i informatike, Varaždin, 2001.
[66] Radošević, D.: "Danijel Radošević - osobna stranica", http://www.foi.hr/~darados/novi/prikaz.cgi, Fakultet organizacije i informatike, 2003., preuzeto 18.08.2004.
[67] Radošević, D.: "On-line vježbe i testovi za programski jezik C++", http://www.foi.hr/~darados/testovi/, Fakultet organizacije i informatike, 2002., preuzeto 18.08.2004.
[68] Ramos-Pollán, R.: "Basic Concepts in Object Oriented Programming", Università degli studi di Firenze, 2002., http://hep.fi.infn.it/JAVAmain.pdf, preuzeto 31.07.2002.
[69] Rideau F.R., Ban J.V.: "Metaprogramming and Free Availability of Sources [] Two Challenges for Computing Today", CNET DTL/ASR (France Telecom), 1999., http:// fare.tunes.org/articles/ll99/mpfas.pdf , preuzeto 29.07.2002.
[70] Rodger, R. J. :"Jostraca: a Template Engine for Generative Programming", 16th European Conference on Object-Oriented Programming, ECOOP’2002, Workshop on Generative Programming, University of Málaga, Spain, June 10-14, 2002.
[71] Rühl S.:"Generative System Programming", http://www-li5.ti.uni-mannheim.de/fpga/gspg/, preuzeto 2.10.2002.
[72] Sells, C.: "Generative programming: Modern Techniques to Automate Repetitive programming Tasks", MSDN Magazine, december 2001, http://msdn.microsoft.com/msdnmag/issues/01/12/GenProg/GenProg.asp, preuzeto 03.10.2002.
[73] Sells, C.: "The reusable generation", ITconsultant, January 2002, http://www.sellsbrothers.com/writing/ITconsultant_column%20inch_Jan2002.pdf, preuzeto 3.10.2002.
[74] Sells, C.:"Generative Programming", http://www.develop.com/conferences/conferencedotnet/materials/N5.pdf, preuzeto 3.10.2002.
[75] Smaragdakis, Y., Batory, D.:"Scoping Constructs for Software Generators", University of Texas at Austin, Department of Computer Sciences,Austin, 1997, http://citeseer.nj.nec.com/cache/papers/cs/1435/ftp:zSzzSzftp.cs.utexas.eduzSzpubzSzpredatorzSzgscope.pdf/smaragdakis97scoping.pdf, preuzeto 7.10.2002.
[76] SEI (Software Engineering Institute):" Domain Engineering: A Model-Based Approach", Carnegie Mellon University, 2002., http://www.sei.cmu.edu/domain-engineering/ domain_engineering.html, preuzeto 18.06.2004.
[77] Stein, D., Hanenberg, S., Unland, R.:"On Representing Join Points in the UML", Proceedings of the Fifth International Conference on the Unified Modeling Language

- the Language and its Applications, Dresden, Germany, 2002., http://lglwww.epfl.ch/workshops/uml2002/papers/stein.pdf, preuzeto 25.06.2004.
[78] Stone J. D.: "Metaprogramming", http://www.math.grin.edu/courses/Scheme/spring-1998/metaprogramming.html, zadnja revizija 21.06.1998., preuzeto 09.11.2001.
[79] Sung, J.J.,:"Aspectual concepts", John Sung's Master Thesis, Northeastern University's College of Computer and Information Science, 2002., http:// www.ccs.neu.edu/research/demeter/DAJ/docs/thesis-sung_2002_05_09_1.doc, preuzeto 24.06.2004.
[80] Štuikys V., Damaševicius R., Ziberkas G., Majauskas G.: "Soft IP Design Framework Using Metaprogramming Techniques", Kaunas University of Technology, 2002., http:// kietas.elen.ktu.lt/~damarobe/ publications/IFIP_2002.pdf, preuzeto 29.07.2002.
[81] Štuikys, V., Damaševičius, R. : " Scripting Language Open PROMOL and its Processor", Software Engineering Department, Kaunas University of Technology, Kaunas, Lithuania, 2000., http://soften.ktu.lt/~damarobe/publications/INFO191.pdf, preuzeto 05.04.2004.
[82] Štuikys, V., Damaševičius, R., Ziberkas, G.: "Open PROMOL: An Experimental Language for Target Program Modification", Software Engineering Department, Kaunas University of Technology, Kaunas, Lithuania, 2001., http://soften.ktu.lt/~damarobe/publications/Vytautas_Stuikys.pdf, preuzeto 5.04.2004.
[83] Van Doren E., "Function Point Analysis", Carnegie Mellon University, Software Engineering Institute., Colorado Springs, 1997.
[84] Völter M, Gärtner A:"Jenerator - Generative Programming for Java", http://www.voelter.de/data/pub/jeneratorPaper.pdf, preuzeto 24.03.2003.
[85] Vranić, V., Navrat, P.:"Multiple Software Development Paradigms and Multi-Paradigm Software Development", Slovak University of Technology, Department of Computer Science and Engineering, Bratislava, 2000.
[86] Wall L., Schwartz R.L.:"Programming Perl", O'Reilly & Associates, Sebastopol, CA, USA, 1991.
[87] Wirth, N. : “Pascal priručnik”, Mikro knjiga, Beograd, 1989.
[88] Yaxin L. : "Programming Language Macro : Grammar", http://www.cc.gatech.edu/classes/cs4410_98_fall/grammar/grammar.htm, preuzeto 07.12.1999.

PRILOG A: POJMOVNIK GENERATIVNOG PROGRAMIRANJA PREMA KONCEPTU TEMELJENOM NA JEZICIMA SKRIPATA

Aktivna biblioteka - predstavlja skup svih predložaka koda (metaskripata) za generiranje aplikacija. Pojedini predlošci pripadaju odgovarajućim generatorima, koji generiraju aplikacije za pojedine problemske domene, odnosno u zadanim programskim jezicima.

Aplikacija - generirani programski kod u ciljnom programskom jeziku, koji je definiran predlošcima koda. Aplikacija može biti generirana pomoću jednog ili više generatora.

Dijagram metaskripata - grafički dijagram ili odgovarajuća tekstualna specifikacija kojom se definira distribucija pojedinih specifičnih svojstava (aspekata), zadanih dijagramom parametara opisa aplikacije, na predloške programskog koda (metaskripte). Dijagram metaskripata sadrži tri tipa elemenata: metaskripta, veza i izvor.

Dijagram parametara opisa aplikacije - grafički dijagram ili odgovarajuća tekstualna specifikacija kojom se definiraju zadani aspekti prema kojima će se pojedina aplikacija razlikovati od ostalih unutar zadane problemske domene. Dijagram parametara opisa aplikacije definira strukturu opisa aplikacije, na temelju kojeg se pomoću odgovarajućeg generatora generira aplikacija.

Funkcije generiranja - izdvojene zajedničke funkcije svih generatora prema skriptnom modelu.

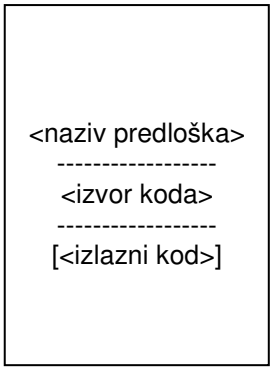
Izvor (eng. source) - predstavlja svojstvo zadana unutar opisa aplikacije, a definirano unutar dijagrama parametara opisa aplikacije, kojim se zamjenjuje odgovarajuća oznaka u dijagramu metaskripata (slika A.1):

Slika A.1: Element izvor u dijagramu metaskripata

Jednorazinski generator - jednostavan generator koji se sastoji od samo jednog predloška i čiji dijagram metaskripata sadrži samo jednu razinu (slika A.1).

Metagenerator - program koji, prema zadanom skriptnom modelu, generira programski kod generatora. Metagenerator sadrži opći model generatora.

Metaskripta (=predložak programskog koda) predstavlja metaprogram, odnosno, program čiji se pojedini dijelovi zadaju pomoću oznaka koje generator zamjenjuje odgovarajućim programskim kodom. Unutar dijagrama metaskripata predložak je predstavljen pravokutnikom (slika A.2).



Slika A.2: Element metaskripta u dijagramu metaskripata

Metaskripte definiraju programski jezik generirane aplikacije, te sintaksu i semantiku programa.

Objektni model - model objektno-orijentirane aplikacije koji definira njene entitete (objekte), njihove atribute i funkcionalnosti, te procese za realizaciju funkcionalnosti.

Opis aplikacije - tekstualna specifikacija pojedine aplikacije unutar njene problemske domene koju koristi generator za generiranje aplikacije. Struktura opisa aplikacije zadana je dijagramom parametara opisa aplikacije.

Oznaka (eng. tag) - predstavlja vezu predloška programskog koda s izvorom podataka, koji je definiran unutar dijagrama parametara opisa aplikacije, a čija specifična vrijednost se zadaje unutar opisa aplikacije, odnosno, predloškom niže razine. Unutar dijagrama metaskripata oznaka predstavlja element veze (slika A.3).

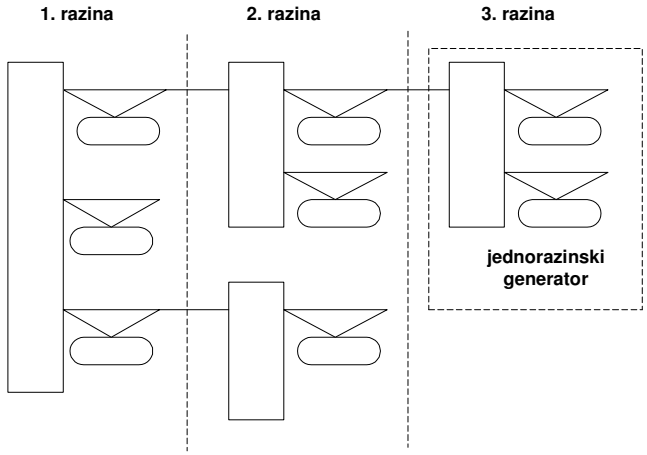
Slika A.3: Element veze u dijagramu metaskripata

Problemska domena (eng. domain) - predstavlja skup predložaka koda čija struktura je zadana dijagramom metaskripata.

Razine generiranja - razlikujemo tri razine generativnog programiranja prema konceptu temeljenom na jezicima skripata:
- razina metageneratora (poopćena razina generatora aplikacija),
- razina generatora aplikacija (poopćena razina aplikacije) .
- razina aplikacije (ciljna razina).

Skriptni model - grafički ili odgovarajući tekstualni model generatora aplikacija prema konceptu temeljenom na jezicima skripata.
Osnovne značajke skriptnog modela u odnosu na objektni su:
- definira generator aplikacija, a ne pojedinu aplikaciju (viša razina)
- orijentiran je na aspekte, a ne na entitete, kao objektni model

Višerazinski generator - složeni generator koji uključuje više predložaka koda na različitim razinama, odnosno, čiji dijagram metaskripata sadrži više razina. Svaka grana u dijagramu metaskripata definira odgovarajući generator niže razine. Najniža razina svake grane dijagrama metaskripata definira jedan jednostavan jednorazinski generator (slika A.4). Višerazinski generator dobije se superpozicijom više jednorazinskih generatora.



Slika A.4: Višerazinski generator dobije se superpozicijom više jednorazinskih generatora.

Višerazinski parametrizirani generator - višerazinski generator realiziran pomoću poziva funkcija generiranja. Parametri funkcija proizlaze iz skriptnog modela generatora.