Matematički fakultet,
Univerzitet u Beogradu

# Problem zadovoljavanja ograničenja pri generisanju ukrštenih reči

Master rad

*Autor:*
Marijana Pešić

*Mentor:*
dr Filip Marić

*Članovi komisije:*
dr Predrag Janičić
dr Saša Malkov

Beograd
2012. godina

# Sadržaj

# Slike

# Glava 1

# Uvod

Ponekad je potrebno izvagati uloženi trud i vreme naspram dobijenih rezultata. Pod naletom novih tehnologija vaga se stalno pomera. Sve manje i manje je potrebno čitati, proučavati i raditi, a rezultati su sve bolji, veći i kvalitetniji. Često problemi na koje nailazimo su već obrađivani u istom ili sličnom obliku. Tako da je teže rešavati problem nego ga prilagoditi nekom drugom problemu ili grupi problema koji su već detaljno obrađivani i zatim iskoristiti tehnike koje su bile korišćene za rešavanje tih problema. To je i razlog što su u ovom radu veći deo zauzeli problemi zadovoljavanja ograničenja (eng. constraint satisfaction problems - CSP), projektni obrasci (eng. design pattern) i slučajevi korišćenja (eng. use cases), koji su najpre objašnjeni i obrađeni korišćenjem više različitih primera, a kasnije su primenjeni i na rešavanje aplikacije za generisanje ukrštenice. Primenom različitih tehnika i njihovom kombinacijom se postiže ubrzavanje aplikacije.

U problemu 8 sahovskih kraljica pretraga ima $\binom{64}{8} = 4426165368$ kombinacija a dobrom postavkom problema ima $8^8 = 16777216$ kombinacija. U aplikaciji za generisanje ukrštenih reči korišćenjem tehnike sa povratnim skokom kojom se obezbeđuje povratak stanja na prethodno stanje koje je narušilo konzistentnost sistema i korišćenjem neke od delimičnih prediktivnih metoda, rad aplikacije se ubrzava.

Da bi se aplikacija približila što većem broju korisnika, ona je postavljena na Internetu. Korišćene su dodatne tehnologije i metode. One su u ovom radu najpre teoretski obrađene, pre svega klijent-server arhitektura, a zatim i primenjene na aplikaciju za generisanja ukrštenih reči.

U dodatku se nalazi terminološki rečnik. Veliki problem prilikom prikupljanja, razumevanja i pisanja materijala je postojanje različitih termina. Odabran je pristup izbegavanja anglicizama i većina termina je korišćena u prevedenom obliku.

# Glava 2

# Pregled postojećih tehnika i alata

## 2.1 Problemi zadovoljavanja ograničenja

Postoje ograničenja svuda oko nas, kao sto su vremenska ograničenja ili materijalna ograničenja. Mi se trudimo da se nosimo sa njima sa različitim uspehom.

„Na koliko načina je moguće postaviti 8 šahovskih kraljica na šahovsku tablu tako da se međusobno ne napadaju?“ tako je glasio problem koji je prvi put objavljen 1848. godine u nemačkom časopisu *Berliner Schachzeitung*. Franc (nem. Franz Nauk) je 1850. godine rešio problem 8 šahovskih kraljica (naveo je 0[1] rešenja). Zanimljivo je da čuveni Gaus (nem. Carl Friedrich Gauß) nije našao sva rešenja, nego samo 72.

Godine 1852. matematičar Fransis Gutri (eng. Francis Guthrie), tada na postdiplomskim studijama u Londonu, bojio je grofovije na karti Engleske. Ubrzo je uočio da mu nije potrebno više od četiri boje. Zapitao se da li su četiri boje dovoljne za svaku kartu i šta je uzrok tome. Pisao je da ne može naći rešenje problema i pokušavao je naći primer u kojem je potrebno pet boja. „Ako mi nađeš primer, mislim da ću morati učiniti isto što i Sfinga.“ govorio je [IG10].

Slične probleme imamo i danas. Ako želimo da napravimo konfiguraciju računara, ograničeni smo novcem, ponudom i onim što želimo da imamo. Sličan je i problem ukrštenih reči. Standardne ukrštene reči su dvodimenzionalne tabele sa redovima i kolonama, poljima u koje treba upisati tražene reči. Po pravilu se u svako polje tabele upisuje po jedno slovo. Reči se na taj način ukrštaju.

Uočimo da ovi problemi imaju nešto zajedničko – objekte kojima se dodeljuju vrednosti: šahovske kraljice koje se postavljaju na šahovska polja, grofovije koje se boje nekom od boja, računarske komponente određene visinom cene i polja ukrštenice u koja se upisuju slova. Kao i kod mnoštva drugih problema možemo izdvojiti objekte, kojima se dodeljuju određene vrednosti i koje nazivamo promenljivima. Takođe, zajedničko je da svaka promenljiva ima određen

[1] Ako se rešenja koja se razlikuju samo po simetriji (rotacija i refleksija) računaju kao jedan, problem ima 12 jedinstvenih rešenja.

skup vrednosti koje joj se dodeljuju. Sve promenljive zadovoljavaju određena ograničenja – šahovske kraljice se međusobno ne napadaju, susedne grofovije nisu iste boje, zbir cena računarskih komponenti zadovoljava količinu novca kojom raspolažemo, presek horizontalne i vertikalne reči je isto slovo.
Postavka problema je sledeća:

– Postoji konačan skup promenljivih
– Za svaku promenljivu postoji konačan skup mogućih vrednosti
– Postoji skup ograničenja koje promenljive moraju zadovoljavati.

Zadatak problema su dodele vrednosti promenljivim iz njihovih domena pri čemu su ograničenja zadovoljena. Problem dodele vrednosti svakoj promenljivoj, od kojih svaka promenljiva uzima vrednost iz svog domena i zadovoljava ograničenja naziva se *problem zadovoljavanja ograničenja (eng. constraint satisfaction problems - CSP)*.

### 2.1.1 Istorijski pregled

Problem zadovoljavanja ograničenja je nastao [Tsa93] u oblasti vestačke inteligencije još 1970. godine. Prvi put je formalizovan za označavanje linija pri vizuelnim istraživanjima od strane Dejvid Hofman (eng. David Huffman [Huf71]) 1971. godine, Maks Kloves (Max Clowes[Clo71]) 1971. godine, Dejvid Valc (eng. David Waltz[Wal75]) 1975. godine i Alan Makvorth (eng. Alan Mackwirth[Mac92]) 1992. godine. Oni su definisali ove tipove problema, sa konačnim domenima, kao probleme zadovoljavanja konačnih ograničenja. Robert Haralik (eng. Robert Haralick[HE80]) i (eng. Stephen Shapiro[SH80]) 1980. godine su dali diskusije na temu različitih prikaza problema zadovoljavanja ograničenja od formalizacije preko aplikacija do algoritma. Mital i Falkenhajner (ind. Sanjay Mittal i Brian Falkenhainer[MF90]) 1990. godine su zamenili standardne probleme zadovoljavanja ograničenja sa dinamičkim problemima zadovoljavanja ograničenja (problemi u kojima ograničenja mogu da budu dodata ili uklonjena). Problem zadovoljavanja ograničenja je prvi put primenjen za univerzitetski raspored 1971. godine [BF71].

Problemi zadovoljavanja ograničenja se izučavaju u oblasti veštačke inteligencije, u teorijskom računarstvu u oblasti teorije složenosti. Matematička logika pruža formalni jezik za opisivanje i analizu problema. Problemi zadovoljavanja ograničenja su često NP-kompletni problemi tj. nije poznat algoritam koji ih može rešiti u polinomijalnom vremenu ali se u polinomijalnom vremenu može pokazati da li je predloženo rešenje zaista rešenje problema. Problem iskazne zadovoljivosti (eng. Boolean satisfiability problem SAT) je problem ispitivanja da li postoje vrednosti iskaznih promenljivih za koje je data iskazna formula (zadata u konjunktivnoj normalnoj formi) tačna [Mar09]. SAT problem je prvi za koji je dokazan da je NP-kompletan [Coo71] i on se može ubrojati u skup problema zadovoljavanja ograničenja. Neki od problema zadovoljavanja ograničenja su: pravljenje vremenskog rasporeda na Habl satalitu [JM94], raspoređivanje vremena sletanja i poletanja na aerodoromu, kriptografija, tumač slika, izvršavanje upita u bazama podataka, gramatika, 8 dama, sudoku, obojivost mapa, iskazna zadovoljivost, mnoštvo drugih počev od problema definisanih na sajtu www.csplib.org ili svetsko takmičenje u rešavanju ovih problema[2].

[2] `http://cpai.ucc.ie/08/`

### 2.1.2 Formalna definicija problema

Pre formalne definicije neformalno ćemo defininisati par pojmova. Svaka promenljiva u problemu zadovoljavanja ograničenja ima domen vrednosti. Domen može biti diskretan ili kontinualan. U diskretnim domenima ograničenja se mogu predstaviti jednostavnom enumeracijom dozvoljenih vrednosti. Ograničenja za kontinualne promenljive zahtevaju algebarske izraze. Ograničenja mogu biti unarna - samo za jednu promenljivu, zatim binarna - koja ograničava uzajamne vrednosti za dve promenljive, ograničenja višeg reda - koja ograničavaju uzajamne vrednosti za tri i više promenljivih. Ograničavanja mogu biti apsolutna, tako da njihovo neispunjavanje potpuno isključuje postojanje mogućeg rešenja ili ograničenja koja ukazuju na najbolje rešenje od više rešenje.

Navešćemo nekoliko formalnih definicija problema zadovoljavanja ograničenja. Definicija problema zadovoljavanja ograničenja prema [RN03]

**Definicija 2.1.1** *Problemi zadovoljavanja ograničenja su definisani skupom promenljivih $X_1, X_2, \ldots, X_n$ i skupom ograničenja $C_1, C_2, \ldots, C_m$. Svaka promenljiva $X_i$ pritom ima neprazan $D_i$ domen mogućih vrednosti.*

Definicija problema zadovoljavanja ograničenja prema [Tsa93]

**Definicija 2.1.2** *Problem zadovoljavanja ograničenja je uređena trojka:*

$$(Z, D, C)$$

*Gde je skup $Z$ konačan skup promenljivih $\{x_1, x_2, \ldots, x_n\}$;*

*$D$ je funkcija koja mapira svaku promenljivu iz skupa $Z$ u skup objekata proizvoljnog tipa:*

$D{:}Z \mapsto$ *konačan skup objekata,*

*$D_{x_i}$ je konačan skup objekata koji se mapira iz $x_i$ pomoću $D$. Ove objekte nazivamo mogućim vrednostima promenljive $x_i$, a $D_{x_i}$ domenom promenljive $x_i$;*

*$C$ je konačan skup ograničenja (može biti i prazan) nad promenljivim iz skupa $Z$. Predstavlja dekartov proizvod domena promenljivih $D_{x_1} \times D_{x_2} \times \ldots \times D_{x_n}$ koje ne mogu biti dodeljene promenljivima $\{x_1, x_2, \ldots, x_n\}$.*

Definicija rešenje problema zadovoljavanja ograničenja prema [Tsa93]

**Definicija 2.1.3** *Rešenje problema zadovoljavanja ograničenja je skup svih vrednosti za koja su zadovoljena ograničenja.*

Prema [Tsa93] problemu zadovoljavanja ograničenja mogu se, u zavisnosti od rešenja, podeliti u tri klase:

– probleme zadovoljavanja ograničenja kod kojih se traže sva rešenja.
– probleme zadovoljavanja ograničenja kod kojih se traže neka rešenja.
– probleme zadovoljavanja kod kojih tražimo optimalna rešenja.

*Binarni problemi zadovoljavanja ograničenja (eng. binary constraint satisfaction problems )* su oni problemi kod kojih ograničenja mogu da budu samo binarna ili unarna tj. da se odnose na jednu ili dve promenljive. Sve probleme zadovoljavanja ograničenja moguće je preformulisati u binarne bez gubitka opštosti, tako da one imaju ista rešenja, što je dokazano u [BW02].

### 2.1.3 Primeri problema zadovoljavanja ograničenja

Prilikom prikaza, šta su i gde postoje problemi zadovoljavanja ograničenja, razmotrićemo neke primere i aplikacije.

**Primer 1.1** Zanimljiv primer za ilustrovanje problema zadovoljavanja ograničenja dat je na početku poglavlja 2.1. Uvedimo promenljive tako da indeks svake promenljive odgovara redu šahovske table na kojoj se nalazi jedna od kraljica, a vrednost promeljive njenoj koloni. Skup ovako definisanih kraljica i promenljivih možemo prikazati na sledeć način:

$$X = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8\}$$

Svaka šahovska kraljica, tj. promenljiva uzima vrednosti iz svog domena. Vrednosti svake kolone označavamo brojevima $1, 2, 3, 4, 5, 6, 7$ i $8$. Domeni za svaku šahovsku kraljicu su

$$D_{x_1} = \ldots = D_{x_8} = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

Ograničenja su da bilo koje dve kraljice ne smeju da se nalaze u istoj koloni (2.1) i bilo koje dve kraljice ne smeju da se napadaju po dijagonali (2.2).

$$C_1 = \{X_i \neq X_j \mid i, j = \overline{1,8} \wedge i \neq j\} \tag{2.1}$$

$$\begin{aligned} C_2 = \{&X_i = x \wedge X_j = y \wedge i - j \neq x - y \wedge i - j \neq y - x \\ &\mid x \in D_{x_i} \wedge y \in D_{x_j} \wedge i, j = \overline{1,8} \wedge i \neq j\} \end{aligned} \tag{2.2}$$

Sa ovakvom postavkom, kardinalnost ovog problema je $8^8 = 16777216$ kobinacija.

**Primer 1.2** Formalizacija 8 šahovskih kraljica može se postaviti i na drugi način. Šahovske kraljice su skup promeljivih

$$X = \{X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8\}.$$

Svaka od promenljivih može da uzme vrednost jednog od $8 \times 8 = 64$ polja šahovske table. Ako polja numerišemo brojevima od 1 do 64 tada je domen za svaku promenljivu

$$D_{x_1} = \ldots = D_{x_8} = \{1, 2, \ldots, 64\}$$

Ograničenja
*Neka je:*

$$R_i = X_i \; div \; 8 + 1, i = \overline{1,8}$$

$$S_i = X_i \; mod \; 8 + 1, i = \overline{1,8}$$

*Tada su ograničenja:*

$$C_1 = \{R_i \neq R_j \mid i, j = \overline{1,8} \wedge i \neq j\}$$

$$C_2 = \{S_i \neq S_j \mid i, j = \overline{1,8} \wedge i \neq j\}$$

$$C_3 = \{R_i - R_j \neq S_i - S_j \mid i, j = \overline{1,8} \wedge i \neq j\}$$

$$C_4 = \{R_i - R_j \neq S_j - S_i \mid i, j = \overline{1,8} \wedge i \neq j\}$$

Definisan na ovaj način problem 8 šahovskih kraljica ima $\binom{64}{8} = 4426165368$ kombinacija.

Uviđamo da isti problem može biti definisan na različite načine, sa različitom efikasnošću. Ista postavka važi za problem $N$ kraljica na šahovskoj tabli $N \times N$, $N \geq 0$.

**Primer 1.3** Za vizualizaciju problema zadovoljavanja ograničenja najčešće se koristi obojivost mapa. Posmatrajmo primer Srbije i njenih susednih država. Postavimo ovaj problem u okviru formalne definicije 2.1.1 problema ograničenja zadovoljavanja. Skup promenljivih su Srbija i susedne države

$$X = \{RS, HU, RO, BG, MK, AL, ME, BA, HR\}$$

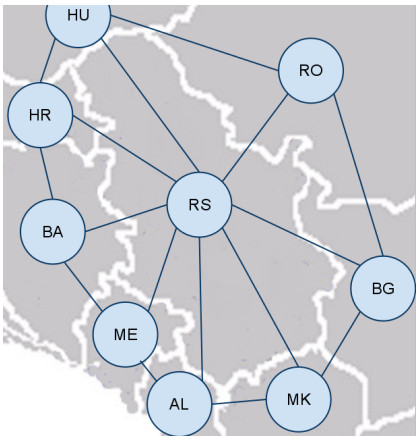
Domen za svaku od promenljivih su tri boje sa kojima bojimo kartu

$$Di = \{crvena, zelena, plava\}, i = \overline{1,9}$$

Ograničenje je da susedne države ne mogu da budu iste boje. Ovaj problem se najlakše predstavlja pomoću grafova. Pre toga uvešćemo nekoliko pojmova.

Graf $G = (V, E)$ predstavlja skup $V$ čvorova i skup $E$ grana. Grana odgovara relaciji između čvorova. Grafički, čvorove predstavljamo krugovima, veze linijama. Stablo je neusmeren acikličan graf, što znači da ne postoji niz povezanih čvorova tako da se jedan čvor ponavlja. Čvor na vrhu stabla naziva se korenom. Čvor bez dece naziva se list. Listovi su završni čvorovi. Čvorovi do kojih vode grane koje polaze iz čvora $X$, nazivaju se deca čvora $X$, a sam čvor $X$ je njihov roditelj.

U problemima zadovoljavanja ograničenja čvorovi odgovaraju promeljivim a lukovi ograničenjima. U primeru obojivosti mape, svaka država predstavlja jedan čvor, koje ćemo obeležiti sa njihovim oznakama. Svake dve susedne države moraju imati različite boje, a to znači da susedni čvorovi imaju lukove (slika 2.1).




Slika 2.1: Mapa Srbije

### 2.1.4 Tehnike rešavanja

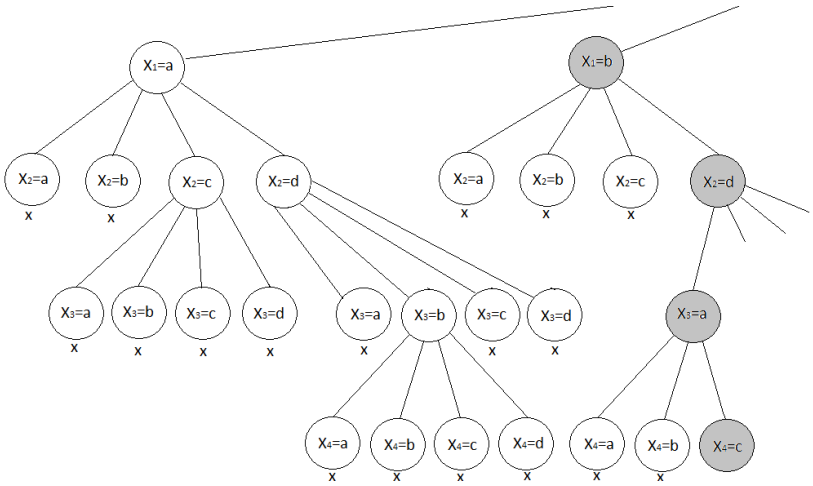
Pošto su domeni promenljivih konačni, moguće je probati sve kombinacije promenljivih i odgovarajućih mogućih vrednosti tj. problem je kombinatorne prirode i može se rešiti pretraživanjem. Broj potencijalnih rešenja tj. broj mogućih dodela vrednosti promenljivama jednak je proizvodu veličina njihovih domena i eksponencijalno raste sa porastom broja promenljivih i/ili njihovih domena. Većina tehnika za rešavanje ovih problema fokusira se na optimizovanju pretrage, redukovanju prostora pretrage u cilju što ranijeg nalaženja rešenja i/ili izboru promenljivih i njihovih vrednosti.

**Tehnike za rešavanje zasnovane na pretraživanju** Najpopularnija tehnika pretraživanja pri rešavanju problema zadovoljavanja ograničenja je *pretraga sa povratkom (eng. backtracking)*. Zasniva se na tome da se održava delimično rešenje (samo nekim promenljivama su dodeljene vrednosti) i inkrementalno se dodeljuje vrednost jednoj po jednoj promenljivoj u svakoj iteraciji. U slučaju da u datoj iteraciji nekoj promenljivoj nije moguće dodeliti bar jednu vrednost iz njenog domena tako da se ne naruše ograničenja, potrebno je vratiti se na predhodnu promeljivu i njoj dodeliti novu vrednost iz skupa nedodeljenih vrednosti. Predstavimo ovo stablom. U njemu čvorovi odgovaraju dodeli vrednosti promenljivoj, grana predstavlja prelazak na dodelu vrednosti sledećoj promenljivoj. Čvor postaje list ukoliko je narušeno neko od ograničenja između njega i roditeljskih čvorova, ili ako se sve ostale promenljive nalaze u roditeljskim čvorovima. Koren ovog stabla je prazan čvor u kojem nijedna promenljiva nema dodeljenu vrednost. Rešenje je uspešno ukoliko su na putu od korena do lista dodeljene vrednosti svim promenljivima i pritom nije narušeno nijedno ograničenje u dodeli vrednosti promenljivoj koja se nalazi u listu.

Posmatrajmo primer četri šahovske kraljice, sa skupom promeljivih $X = \{X_1, \ldots, X_4\}$, domenom za svaku promenljivu $D_{x_1} = \ldots = D_{x_4} = \{a, b, c, d\}$ i ograničenjem da se kraljice ne napadaju. Na slici dela stabla pretrage (slika 2.2) možemo uočiti da vrednost $a$ je dodeljena prvoj kraljici, zatim se drugoj kraljici dodeljuje vrednost $a$, gde se završava ova grana pretrage jer postoji konflikt tj. nije zadovoljeno ograničenje, prva i druga kraljica se napadaju. Drugoj kraljici se dodeljuje nova vrednost $b$ i takođe se završava pretraga jer postoji konflikt (konflikt je na crtežu označen sa x ispod lista). Drugoj kraljici je dodeljena vrednost $c$, zadovoljena su ograničenja. Nakon što su sve vrednosti za treću kraljicu iscrpljene pretraga se vraća na kraljicu dva i njoj se dodeljuje vrednost $d$. Ograničenja su zadovoljena i trećoj kraljici je dodeljena nova vrednost i tako nastavljamo dalje algoritam.

Ovaj osnovni oblik algoritma ima nekoliko mana. Prva je da se pri vraćanju uz stablo ne vraća do promenljive koja je izazvala konflikt pa se bespotrebno proveravaju kombinacije koje dovode do istog konflikta. Druga je to što se pri detektovanju konflikta on ne pamti pa je moguće da se više puta probaju kombinacije koje izazivaju isti konflikt. Treća je da se konflikti otkrivaju suviše kasno.

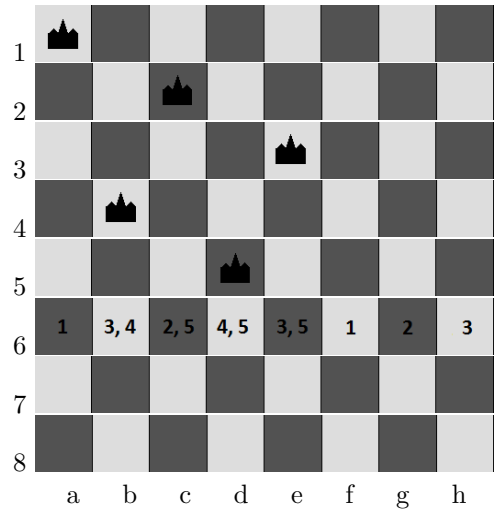
Prva mana se rešava proširivanjem algoritma sa tehnikom *povratni skokovi (eng. backjumping - BJ)*. Algoritam je sličan pretrazi sa povratkom osim što se u slučaju pronalaženja konflikta pokušava na osnovu narušenog ograničenja da pronađe izvor problema i vrati se uz stablo do izvora konflikta tj. prilikom povratka u pretrazi se ponište nekoliko dodela odjednom sve do poslednje dodele




Slika 2.2: Deo stabla pretrage za šahovske kraljice

koja je učestvovala u konfliktu.

Razmatrimo primer osam šahovskih kraljica, sa skupom promeljivih $X = \{X_1, \ldots, X_8\}$, domenom za svaku promenljivu $D_{x_1} = \ldots = D_{x_8} = \{a, \ldots, h\}$ i ograničenjem da se kraljice ne napadaju (slika 2.3). Posle dodele vrednosti petoj kraljici, svaka vrednost koja se dodeli šestoj kraljici stvara konflikt. Na slici su prikazani brojevi kraljica koji stvaraju konflikt pri dodeli šestoj kraljici. Pretraga sa povratkom[3] bespotrebno proverava sve kombinacije za petu kraljicu koje dovode do istog konflikta, dok se pretraga sa povratnim skokom vraća do četvrte kraljice i nastavlja se sa dodelom.




Slika 2.3: Šahovske kraljice – povratni skokovi

[3]Aplikacija koje ilusturuje pretragu sa povratkom `http://alas.matf.bg.ac.rs/~mr03106/queens_puzzle/queens_puzzle.html` – urađena koristeći Procesing

Pratimo nekoliko koraka:

| | | | |
|---|---|---|---|
| $kraljica$ | 1 | $a$ | |
| $kraljica$ | 2 | $abc$ | |
| $kraljica$ | 3 | $abcde$ | |
| $kraljica$ | 4 | $ab$ | |
| $kraljica$ | 5 | $abcd$ | |
| $kraljica$ | 6 | $abcdefgh$ | (neuspeh, skok na kraljicu 4) |
| $kraljica$ | 4 | $cdefg$ | |
| $kraljica$ | 5 | $a$ | |
| $kraljica$ | 6 | $ab$ | |

...

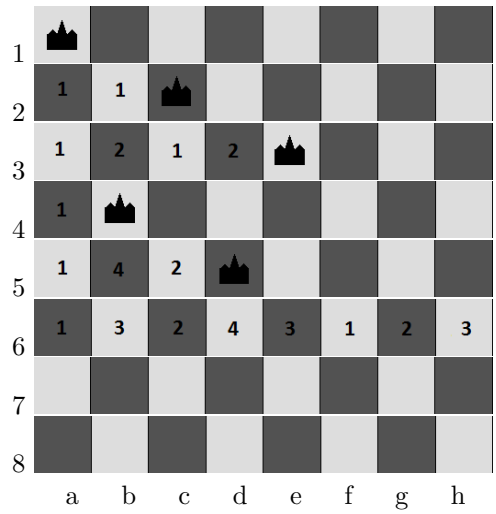
Sledeće poboljšanje pretrage sa povratkom je *markiranje (eng. backmarking - BM)* koje pokušava da otkrije zavisnosti između promenljivih na različitim nivoima stabla pretrage i tako izbegne ponavljanje istih konflikata. Neka su promenljive u redosledu $X_1, X_2, .., X_n$. Pri prolasku kroz stablo pretrage markirani su najmanji indeksi kraljica za polja kojima nije moguće dodeliti promenljivu jer se narušavaju ograničenja. Ako se algoritam vrati na kraljicu i dodeli joj se nova vrednost, brišu se markirana polja u kojima je ta kraljica učestvovala. Pretpostavimo da nijedna dodela promenljivoj $X_j$ ne zadovoljava uslove i da moramo odustati od svih promenljivih do $X_i$, gde je $i < j$. Kada promenljivoj $X_i$ dodelimo novu vrednost ne proveravaju se markirana polja jer je za njih već utvrđeno da izazivaju konflikt.

Razmatrimo primer za osam šahovskih kraljica (slika 2.4). Nakon svake dodele vrednosti od prve do pete kraljice za polja koja su izazivala konflikt izračunat je najmanji indeks kraljice koja stvara konflikt i markiran kao na slici 4. Utvrđeno je da nema vrednosti za šestu kraljicu, algoritam se vraća na kraljicu pet. Proverava se polje $e$, koje narušava ograničenja sa kraljicom jedan i kraljicom tri. Polje se markira sa indeksom 1, polje $f$ sa indeksom 2, polje $g$ sa indeksom 3. Nakon dodeljivanja polja $h$ petoj kraljici, algoritam prelazi na kraljicu šest. Sva polja su markirana. Algoritam se vraća na kraljicu četiri. Brišu se markeri u kojima učestvuje kraljica četiri. Kraljici četiri se dodeljuje vrednost $g$. Markiraju se polja koja narušavaju konflikt. Algoritam prelazi na kraljicu pet, preskače markirano polje i proverava polje $d$ za kraljicu pet.

Pratimo nekoliko koraka:

| | | | |
|---|---|---|---|
| $kraljica$ | 1 | $a$ | |
| $kraljica$ | 2 | $abc$ | markiraju se vrednosti a(1), b(1) |
| $kraljica$ | 3 | $abcde$ | markiraju se vrednosti a(1), b(2), c(1), d(2) |
| $kraljica$ | 4 | $ab$ | markira se vrednost a(1) |
| $kraljica$ | 5 | $abcd$ | markiraju se vrednost a(1), b(4), c(2) |
| $kraljica$ | 6 | $abcdefgh$ | (neuspeh, natrag na kraljicu 5) markiraju se vrednosti a(1), b(3), c(2), d(4), e(3), f(1), g(2), h(3) |
| $kraljica$ | 5 | $efgh$ | markiraju se vrednosti e(1), f(2), g(2) |
| $kraljica$ | 6 | | (neuspeh, natrag na kraljicu 5) sva polja su markirana |
| $kraljica$ | 5 | | (neuspeh, natrag na kraljicu 4) sve vrednosti su isprobane |
| $kraljica$ | 4 | $cdefg$ | markiraju se vrednost c(2), d(1), e(3), f(3) |
| $kraljica$ | 5 | $bdh$ | |

...



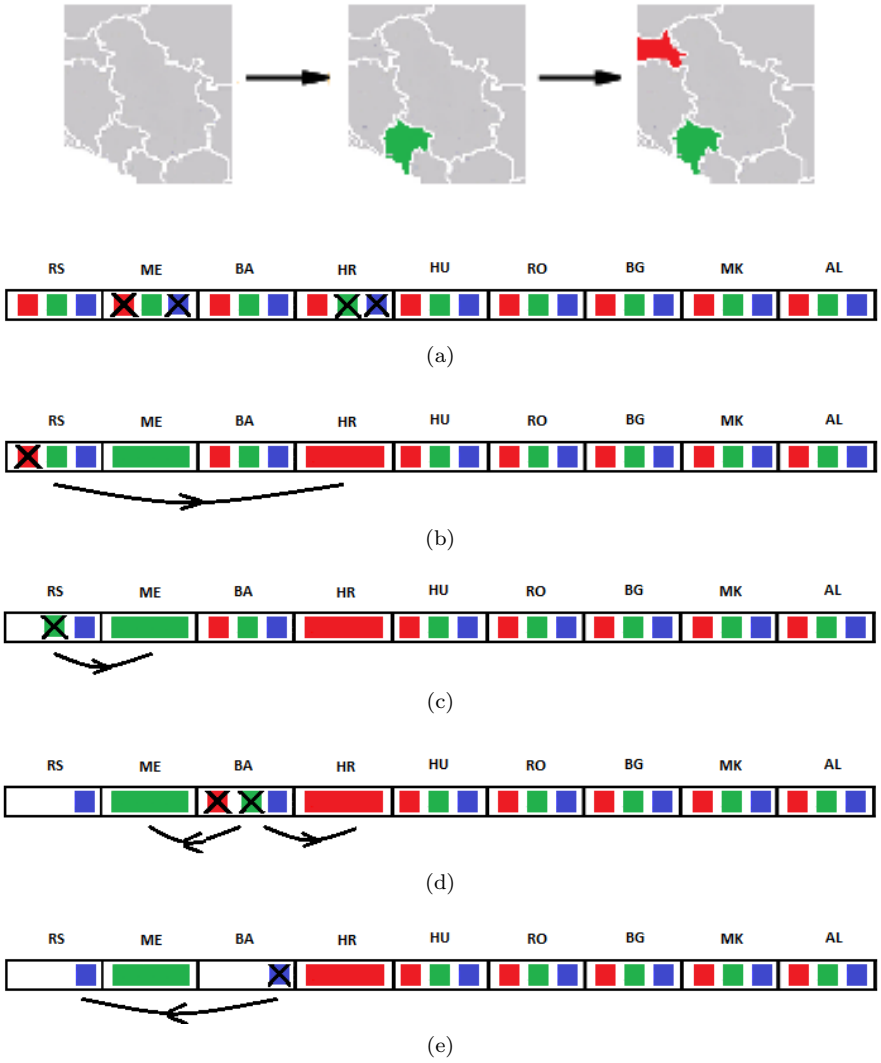

Slika 2.4: Šahovske kraljice – markiranje

**Tehnike očuvanja konzistentnosti** Tehnike očuvanja konzistentnosti fokusiraju se na što ranijem otkrivanju narušavanja ograničenja pomoću restrikcije domena. Skup ograničenja se može predstaviti grafom čiji čvorovi su promenljive a grane predstavljaju ograničenja. Jedna grana predstavlja jedno binarno ograničenje (ograničenje između dve promenljive). Ovo ne umanjuje opštost pošto se svako ograničenje stepena većeg od dva može predstaviti pomoću više binarnih ograničenja.

Najprostija tehnika je *konzistentnost čvorova (eng. node consistency - NC)*. Pri svakoj dodeli vrednosti nekoj promenljivoj redukuju se domeni ostalih promenljivih. Čvor u grafu ograničenja $X$ je konzistentan čvor ako i samo ako su za svaku vrednost iz trenutnog domena $X$ zadovoljena sva unarna ograničenja promenljive $X$. Svaki čvor u svakom trenutku mora da zadovolji uslove.

Korak dalje ide tehnika *konzistentnost luka (eng. arc consistency - AC)*. Kod ove tehnike u svakom trenutku mora biti ispunjen uslov da za svaku promenljivu $X$ i za svaku vrednost iz njenog trenutnog domena postoje vrednosti ostalih promenljivih tako da ni jedno ograničenje nije narušeno.

U problemu 2.1.3 bojenja mape Srbije i susednih država koji je ranije definisan dodata su još dva unarna ograničenja. $ME$ je zelene boje i $HR$ je crvene boje. Na slici 2.5a proverena su samo unarna ograničenja i na njoj su svi čvorovi konzistentni. Na slici 2.5b proverava se konzistentnost luka, ako promenljiva $RS$ uzme vrednost crvena boja, tada nije ispunjeno ograničenje da susedne zemlje nemaju istu boje, jer u domenu za HR ne postoji takva vrednost da je zadovoljeno ograničenje. Izbacuje se crvena boja iz skupa domena promenljive $RS$. Slika 2.5c proverava zadovoljivost luka ako $RS$ uzme vrednost zelena boja. Ograničenje nije zadovoljeno jer u domenu za $ME$ ne postoji takva vrednost da je zadovoljeno ograničenje. Izbacujemo zelenu boju iz skupa domena promenljive $RS$. Na slici 2.5d proveravamo zadovoljivost vrednosti domena za promenljivu $BA$ i izbacujemo crvenu i zelenu boju iz domena $BA$. Na slici 2.5e proveravamo zadovoljivost luka ako $BA$ uzme jedinu preostalu vrednost – plavu boju. U domenu $RS$ ne postoji promenljiva koja zadovoljava ograničenje da




Slika 2.5: Mapa Srbije – tehnike očuvanja konzistentnosti

susedne zemlje nemaju istu boje. Skup promenljivih za $BA$ je prazan i luk nije zadovoljiv. Za problem obojivosti mape Srbije i susednih država sa tri boje sa dodatim unarnim ograničenjima ne postoji rešenje[4].

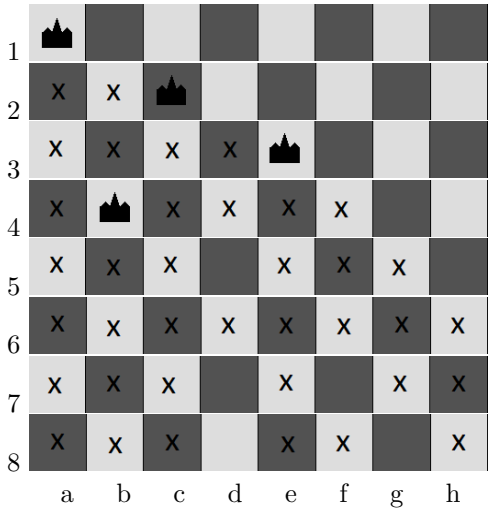
Sledeća tehnika očuvanja konzistentnosti je *konzistentnost puta (eng. path consistency - PC)* i ona potpuno eliminiše potrebu za pretragom. Kod ove tehnike u svakom trenutku mora biti ispunjen uslov da za svaki par promenljivih $X$ i $Y$ i za svaku vrednost iz njihovih trenutnih domena postoje vrednosti ostalih promenljivih tako da ni jedno ograničenje promenljive $X$ sa ostalim vrednostima i promenljive $Y$ sa ostalim vrednostima nije narušeno. Ime konzistentan put potiče od originalne definicije, koja je uključivala promenjive i put (eng. path) između njih, a ne par promenljivih [MS98].

[4]Po uzoru na primer [Cha]

Konzistentan čvor može se nazvati i *jednostruko konzistentan (eng. 1-consistency)*, a konzistentnost luka - *dvostruko konzistentan (eng. 2-consistency)* prema [Tsa93]. Posmatramo $k$ promenljivih. Ako za svaku vrednost iz njihovih trenutnih domena postoje vrednosti ostalih promenljivih tako da ni jedno ograničenje $k$ promenljive sa ostalim vrednostima nije narušeno nazivamo onda - *k- konzistentan (eng. k-consistency)*, gde je $k$ manje ili jednako od broja promenljivih. Prema [Tsa93] moguće je izračunati $k$ tako da se potpuno eliminiše potreba za pretragom. Implementacija ove tehnike je vremenski jako zahtevna tako da se ona retko koristi.

**Prediktivne tehnike** Tehnike koje su kombinacija algoritma za pretraživanje sa povratkom i neke od tehnika očuvanja konzistentnosti nazivaju se *prediktivne tehnike (eng. look ahead)*. Najprostija je *proveravanje unapred (eng. forward checking - FC)*. Sastoji se u sledećem: pri izboru vrednosti za dodelu promenljivoj $X$, moraju postojati vrednosti za sve ostale promenljive koje se pojavljuju u ograničenjima zajedno sa $X$ tako da ova ograničenja budu zadovoljena. Ovaj dodak omogućava pretrazi sa povratkom da ranije otkrije konflikte i izbegne nepotrebne dodele vrednosti.

Posmatramo primer osam šahovskih kraljica (slika 2.6). Nakon izbora prve kraljice, domeni ostalih kraljica se redukuju i neprazni su. Na isti način stiže se do četvrte kraljice. Nakon redukovanja domena ostalih kraljica, može se uočiti da je domen za šestu kraljicu redukovan na prazan skup. Nakon toga algoritam se vraća na poslednju dodelu, dodeljuje drugu vrednost, redukuje domen za sve kraljice sa indeksom većim ili jednakim od četiri i nastavlja se dalje algoritam pretrage.



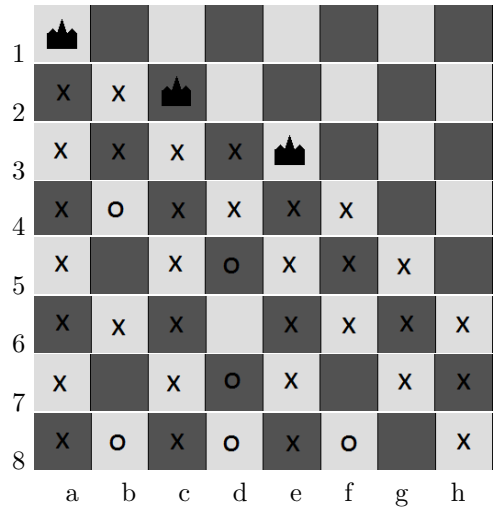

Slika 2.6: Šahovske kraljice – proveravanje unapred

Pratimo nekoliko koraka:

| | | |
|---|---|---|
| *kraljica* | 1 | *a* |
| *kraljica* | 2 | *c* |
| *kraljica* | 3 | *e* |
| *kraljica* | 4 | *bh* |
| *kraljica* | 5 | *b* |
| *kraljica* | 6 | *d* |
| | ... | |

Nešto složenije tehnike su *delimična i potpuna prediktivna tehnika (eng. partial look ahead and full look ahead)*. Za razliku od proveravanja unapred, ove tehnike idu dalje u proveravanju konzistentnosti tako da delimična prediktivna tehnika ne proverava samo ,,susedne" promenljive promenljivoj koja se trenutno instancira, već i njihove susedne promenljive ili čak i više koraka unapred, dok poptuna prediktivna tehnika u obzir uzima sve promenljive kojima još nije dodeljena vrednost. Što se više nivoa proverava, troši se više vremena na proveru a manje na samoj pretrazi tako da je, zavisno od konkretnog problema, potrebno napraviti kompromis tako da implementacija provere konzistentnosti ne izazove čak lošije performanse u odnosu na originalni algoritam.

U algoritmu, koji je spoj *algoritma za pretraživanje i tehnika konzistentan put (eng. arc consistency look ahead)*, pri izboru vrednosti za dodelu promenljivoj $X$ moraju postojati vrednosti za sve ostale promenljive koje se pojavljuju u ograničenjima zajedno sa $X$, tako da ova ograničenja budu zadovoljena. Provera koja važi za $X$ važi i za sve promenljive koje se pojavljuju u ograničenjima sa $X$. Primer delimične prediktivne motode posmatramo na primeru 8 kraljica (slika 2.7). Nakon izbora vrednost za treću kraljicu na slici su sa ,,x" označeni redukovani domeni zbog jednostruke konzistentnost a sa ,,o" redukovani domeni zbog dvostruke konzistentnosti.




Slika 2.7: Šahovske kraljice – delimična prediktivna tehnika

Pratimo nekoliko koraka:

| | | |
|---|---|---|
| *kraljica* | 1 | *a* |
| *kraljica* | 2 | *c* |
| *kraljica* | 3 | *e* |
| *kraljica* | 4 | *gh* |
| *kraljica* | 5 | *b* |
| *kraljica* | 3 | *f* |
| *kraljica* | 4 | *bh* |
| *kraljica* | 3 | *g* |
| ... | | |

**Heuristike izbora promenljivih i njihovih vrednosti** Redosled instanciranja promenljivih značajno utiče na performanse pretrage sa povratkom. Redosled može biti statički i dinamički. Statički redosled je utvrđen pre početka pretrage i ne menja se u toku pretraživanja. Dinamički redosled nije fiksiran na početku pretrage već se izbor promenljive vrši u svakoj iteraciji na osnovu podataka koji su poznati tek u vreme pretrage. Kod osnovnog oblika pretrage sa povratkom dinamički redosled nije ni moguć jer ne postoje podaci koji bi se mogli iskoristiti za dinamički izbor promenljivih. Provera konzistentnosti uključuje izračunavanje domena preostalih promenljivih ili bar nekih od njih tako da se ovi podaci mogu iskoristiti za dinamički izbor sledeće promenljive za instanciranje. Postoji više heuristika za dinamički izbor promenljivih a najčešće korišćena (i jedna od najefikasnijih) je popuniti *promenljivu sa najmanjim domenom (eng. fail first ili minimum remaining value - MRV)*. Ova heuristika za instanciranje bira promenljivu sa najmanjim preostalim domenom. Instanciranje promenljive redukuje domene nekih ili svih preostalih promenljivih tako da je verovatnije da će se do rešenja doći ako se promenljive sa najvećim preostalim domenima instanciraju što kasnije. Pošto se pri implementaciji tehnika očuvanja konzistentnosti već izračunavaju domeni preostalih promenljivih, ova heuristika je prirodan izbor za algoritme koji se oslanjaju na tehnike očuvanja konzistentnosti.

Kada je izabrana promenljiva za instanciranje potrebno je izabrati jednu od mogućih vrednosti iz trenutnog domena date promenljive. Izbor vrednosti takođe značajno utiče na performanse algoritma. Potrebno je izabrati vrednost koja će najverovatnije dovesti do rešenja. Postoji nekoliko heuristika koje se koriste pri izboru vrednosti. Najjednostavniji način je izbor one vrednosti koja ostavlja najviše opcija tj. koja najmanje redukuje domene preostalih promenljivih. Druga heuristika bi bila izbor one vrednosti koja ostavlja sistem u stanju u kome je najlakše rešiv. Ovo podrazumeva postojanje funkcije koja izračunava ,,težinu rešivosti" sistema (npr. na osnovu grafa ograničenja). Implementacija ovih heuristika često je vremenski ,,skupa", naročito kod promenljivih sa velikim domenima tako da ih treba koristiti sa oprezom.

Zanimljiv algoritam koji se koristio za rešavanje 1024 kraljice ili koji je problem pravljenja vremenskog rasporeda na Hablu smanjio sa 7 dana na 10 minuta je *algoritam minimalnih konflikta (eng. min-conflicts algorithm)*. Algoritam dodeljuje slučajnu vrednost za sve promenljive. Zatim se nasumično izabere promenljiva koja narušava ograničenja i dodeljuje joj se vrednost sa minimalnim brojem narušavanja ograničenja. Ako postoji više od jednog minimuma onda se vrednost bira slučajno. Zatim iteracija počinje iznova dok se ne nađe izlaz ili ne postigne maksimalan broj iteracija.

## 2.2 Klijent-server arhitektura

Arhitekture mreža tipa klijent-server postale su popularne krajem 1980-ih i početkom 1990-ih godina, kada su mnoge aplikacije prešle sa centralizovanih računara na mreže personalnih računara kao rezultat nastojanja da se što bolje iskoriste resursi. Klijent-server model je baziran na distribuciji funkcija između dva tipa autonomnih procesa: servera i klijenta. Klijent je proces koji zahteva usluge od serverovog procesa. Server je proces koji obezbeđuje servise za klijente. Mogu biti smešteni na istom računaru ili različitim računarima povezanim preko mreže. Relacija između klijenta i servera je $M : N$, gde jedan server može da opsluži više klijenata a sa druge strane jedan klijent može koristiti usluge više servera. Tako da elementi koji čine klijent-server model su: jedan ili više klijenata, jedan ili više servera, klijentski procesi, serverski procesi i komunikacijski međusloj (komunikacijski hardver i softver). Tipičan scenario po kome radi klijent-server model je da se server proces inicijalizuje, pokreće, a zatim prelazi u mod mirovanja i čeka da ga neki klijent proces kontaktira i zatraži neku uslugu od njega. Klijent proces šalje zahtev putem mreže do servera tražeći određenu uslugu od njega. Kada server proces završi servis koji je od njega zatražen od strane klijenta, prelazi ponovo u mod mirovanja i čeka sledeći zahtev za nekom uslugom. Komunikacioni posrednik između klijenta i servera je povezan sa mrežom. Svi klijentovi zahtevi i odgovori servera putuju kroz mrežu u obliku poruke koja se sastoji od informacija za kontrolu i prenos podataka.

Klijent sadrži hardverske i softverske komponente i poželjno je da one poseduju operativni sistem koji je sposoban da podrži multitasking i komunikacijske sposobnosti. Klijent aplikacija se spaja sa operativnim sistemom radi korišćenja multitaskinga i grafičkog korisničkog interfejsa koje on osigurava. Ona se još spaja i sa mrežnom softverskom komponentom komunikacijskog posrednika radi pristupa servisima. Server poseduje softversku i hardversku komponentu. Računari koji rade kao server najčešće moraju biti mnogo snažniji od uobičajenih klijent računara zato što server procesi moraju da zadovolje konkurentne zahteve više klijenata. Sistem zasnovan na klijent-server modelu je veoma fleksibilan i otvoren za sve vrste izmena hardvera i softvera.

Klijent i server procesi moraju biti autonomni, sa jasno definisanim granicama i funkcijama. Ova osobina omogućava jasno definisanje funkcionalnosti obe strane. Svi principi moraju biti bazirani na standardima koji se koriste unutar klijent-server modela. Univerzalni standardi još uvek ne postoje već ima mnogo različitih standarda koji se mogu primeniti na upravljanje korisničkim interfejsom, pristup podacima, međuprenosnu komunikaciju.

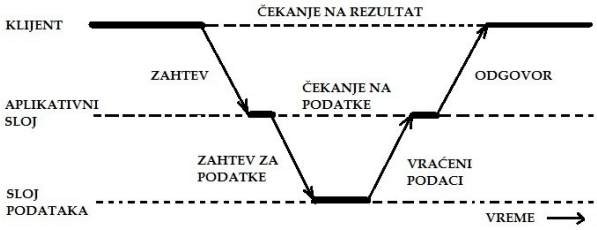
### 2.2.1 Veb klijent-server aplikacija

Veb (eng. World Wide Web - WWW) je najznačajniji Internet servis i baziran je na klijent-server modelu. Sačinjavaju ga veb stranice koje su smeštene na Veb serverima i na zahtev klijenata se prenose na klijentski računar i prikazuju kroz pregledač Veba (eng. Web browsers). Veb stranice zajedno sa specijalizovanim Veb serverima nazvaćemo Veb aplikacija (eng. Web Application)[5]. Veb aplikacija može biti sastavljena iz slojeva. Razlikujemo jednoslojne, dvoslojene,

[5]Pojam je uveden 1999. godine sa Javom 2.2 i Servletima

troslojne i višeslojne aplikacije u zavisnosti gde se nalaze podaci i poslovna logika. Kod dvoslojnih aplikacija klijent pokreće aplikaciju koja preko mreže pristupa serveru baze podataka. Zatim izvršava logiku i prikazuje tražene rezultate na ekranu. Ovakav oblik aplikacije je pogodan za sisteme sa malim brojem korisnika, teški su za održavanje i sigurnost podataka.

Danas su češće troslojne i višeslojne aplikacije. Troslojnu aplikacija čine tri sloja. Prvi sloj je *prezentacioni sloj (eng. presentation tier)* koji predstavlja klijenta. Srednji sloj je *aplikacioni sloj (eng. application tier)* i treći sloj predstavlja *sloj podataka (eng. data tier)*. Prvi sloj šalje zahtev za nekom stranicom serveru. Aplikacijski sloj obrađuje zahteve, komunicira sa bazom podataka i dohvata podatke, zatim ih obrađuje, pa ih onda šalje natrag klijentu (slika 2.8).




Slika 2.8: Troslojna aplikacija - tok

Definicija troslojne arhitekture [Vla09, Sch95]

**Definicija 2.2.1** *Troslojna arhitektura se sastoji iz sledećih slojeva:*

1. *Prezentacionog sloja koji predstavlja ulazno – izlaznu reprezentaciju softverskog sistema.*
2. *Aplikacionog sloja koji opisuje strukturu i ponašanje softverskog sistema.*
3. *Sloja podataka koji čuva stanje atributa softverskog sistema.*

Sloj koji je vidljiv samom korisniku, naziva se i *korisnicki interfejs (eng. user interface)*. On prihvata podatke koje unosi korisnik, prihvata događaje koje pravi korisnik, prosleđuje i prihvata podatke od aplikacionog sloja. Prezentacioni sloj je nezavisan od aplikacionog sloja i sloja podataka. Omogućavaju da se za isti aplikacioni sloj vežu različiti prezentacioni slojevi.

Aplikacioni sloj sadrži aplikacije koje su zadužene za: deo za komunikaciju sa prezentacionim slojem, deo za komunikaciju sa slojem podataka i poslovnu logiku. Poslovna logika je opisana sa strukturom (domenskim klasama) i ponašanjem (sistemskim operacijama)[Vla09]. U srednjem sloju danas se zbog fleksibilnosti i prenosivosti na različite platforme često koriste i razni skript jezici.

Sloj podataka sadrži sistem za upravljanje bazom podataka, kao i sama baza podataka. Baza podataka se po pravilu nalazi na odvojenom mestu od aplikacionog sloja. Ovaj sloj služi za dodavanje, promenu i dohvatanje podataka. Omogućava pristup podacima sa više odvojenih servisa, osigurava sigurnost, tajnost i integritet podataka. Danas postoji više vrsta baza podataka. Jedan od najpopularnijih, zbog jednostavnosti, otvorenog koda i podrške na više platformi, je $MySQL$.

Na osnovu troslojne arhitekture su napravljeni savremeni *aplikacioni serveri (eng. Application servers)*. Aplikacioni server je okruženje koje izvršava aplikaciju. Termin aplikacioni server prvobitno je korišćen za softver koji je veza između klijent-server modela i baze podataka i sistema. Kasnije, paralelno sa pojavom termina Veb aplikacija, aplikacioni server dobija mnogo više funkcionalnosti, slojeva, podršku za distribuirane sisteme, klastere (eng. cluster), korisnički interfejs ka sistemu. Termin *Veb server (eng. Web server)* se koristi za okruženje koje izvršava aplikaciju. Nalazi na istoj mašini kao i sama aplikacija i sluzi za dinamičko izvršavanje stranica i vezu ka bazi...

## 2.3 Korišćeni alati i tehnologije

**Server - Java EE** *Java enterprajz izdanje (eng. Java Enterprise Edition - Java EE)*[6] obuhvata *aplikacioni programski interfejs (eng. Application programming interface - API)* koji podržava složene, višeslojne desktop i Veb aplikacije. Korišćenjem ovog aplikacionog programskog interfejsa razvijaju se distribuisane[7] klijent-server aplikacije u Javi takve da mogu da se izvršavaju na različitim operativnim sistemima ili hardverskim platformama. Osnovnu arhitekturu Jave EE čine servleti, JSP strane (eng. JavaServer Pages) i EJB (eng. Enterprise JavaBean). Servleti i JSP se koriste za komunikaciju sa klijentom. U EJB se smešta poslovna logika i komunikacija sa bazom, tj. sve ono što se ne nalazi na aplikativnom serveru . Sam aplikativni server se sastoji od udaljenog i tekućeg interfejsa (eng. home and remote inerface), EJB, konteksta i kontejnera. Komunikaciju započinje klijent pozivanjem udaljene metode. Kontejner prima, ako je u skladu sa onim što je definisano, prosleđuje poziv odgovarajućem EJB. EJB komunicira sa bazom, a komunikacija može da se izvršava preko nekog kontejner elemanta. Standardni interfejs za pristup SQL bazama podataka je JDBC (eng. Java Database Connectivity). Java IDL (eng. Java Interface Definition Language) je razvijen kao način specificiranja interfejsa između objekata i njihovih klijenata (koji su na razlicitim platformama) na način nezavistan od jezika. Java RMI (eng. Java Remote Method Invocation) omogućava pozivanje udaljenih metoda između klijenta i servera u slučajevima kad se na oba kraja poziva nalaze aplikacije pisane u jeziku Java. Bitan element svake Java EE Veb aplikacije je XML dokument, opisivač rasporeda (eng. deployment descriptor), koji opisuje podešavanja rasporeda aplikacije.

**Klijent - Fleks** Fleks (eng. Flex) je visoko produktivan, besplatan alat koji se koristi za izradu naprednih Internet aplikacija (Rich Internet Application - RIA). Fleks je prvi put objavljen od strane Makromedija (eng. Macromedia) u martu 2004. godine. U Fleksu Adobe vidi priliku i za daljim razvojem Fleša (eng. Flash). Fleš je u početku bio namjenjen dizajnerima i izradi animacija i

[6] Bila je poznata kako J2EE (eng. Java 2 Platform, Enterprise Edition), nakon verzije 5 promenjeno je ime u Java EE. Trenutna verzija se naziva Java EE 6

[7] Prema [Wika] reč *distribuirani (eng. distributed)* u izrazima kao što su ,,distribuirani sistem", ,,distribuirano programiranje", i ,,distribuirani algoritam" u originalu se odnose na računarske mreže gde su samostalni računari distribuirani na određenom geografskom području komuniciraju putem računarske mreže radi postizanja zajedničkog cilj.[Lyn96]. U novije vreme se koristi u mnogo širem značenju, čak i kada se odnosi na autonomne računarske procese koji se odvijaju u samom računaru kao jedinci, i koji međusobno deluje sa drugima razmenjivanjem poruka [And00].

vizuelnih efekata. Sada je Fleš tehnologija prisutna na Internetu i upotrebljava se u mnoge svrhe: izradu Veb stranica, oglašavanje, izradu prezentacija, prikaz multimedijskih sadržaja, video... Programerima standardnih aplikacija stvaralo je poteškoće da se prilagode animacijama na kojima je originalno bazirana Fleš platforma. Fleks minimizira ove poteškoće time što obezbeđuje tok radnje i model programiranja koji je približniji programerima. Interakcija je postignuta korišćenjem ActionScript, osnovnom jeziku Adobe Fleš plejera (eng. Adobe Flash Player) koji je zasnovan na ECMA skript (eng. ECMA Script) standaradu.

Internet aplikacije napravljene pomoću Fleksa mogu se pokretati pomoću Adobe Fleš plejera unutar Veb pretraživača. Adobe Fleš plejer je pomoćni program koji služi za pokretanje .swf i .flv datoteka. Prema tvrdnjama Adobe organizacije 93% korisnika Interneta ima instalirano Adobe Fleš plejer na računaru. Velika prednost Adobe Fleš plejera je nezavisnost od platforme na kojoj se koristi. Na samom početku razvoja, Adobe Fleš plejer je bio dizajniran za prikazivanje dvodimenzionalnih vektorskih animacija. Danas je stigao do naprednih Internet aplikacija, kao i videa i audia. Adobe Fleš plejer 10 omogućava 3D transformaciju i animaciju, naprednu audio obradu, te ubrzanje GPU hardvera i vrlo fleksibilni tekstualni mehanizam. Termin napredne Internet aplikacije - RIA je po prvi put upotrebljen ,,davne" 2002. godine od strane inženjera Makromedije (eng. Macromedie) (danas Adobe) koji su odlučili da razvijaju na novom konceptu Veba i desktop aplikacija koje će po rečima Adobe tima pružiti ,,napredno korisničko iskustvo koje povećava produktivnost i zadovoljstvo aplikacijom".

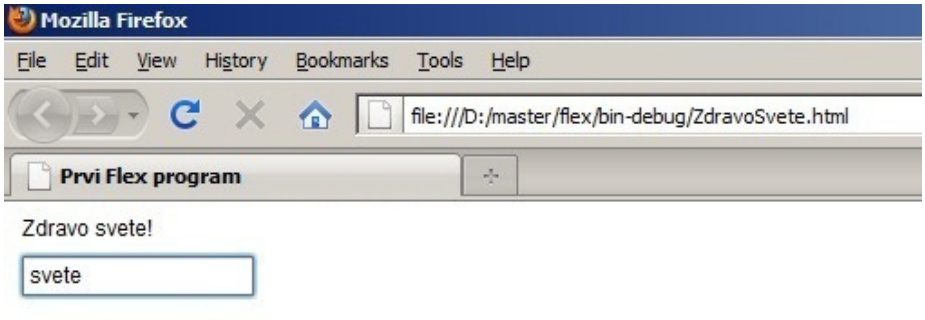
Za izradu aplikacija, Fleks koristi dva jezika: MXML i ActionScript. MXML je jezik koji je baziran na jeziku XML i pridržava se određenih pravila XML jezika za označavanje podataka. MXML je deklaracijski jezik koji se koristi za opis rasporeda komponenata i njihovo ponašanje. Jezik koji radi u pozadini je ActionScript. ActionScript je skript jezik čiji je cilj kontrola objekata, kreiranje navigacionih i interaktivnih elemenata i za kreiranje filmova visokog stepena interakcije. Sintaksa podseća na Javu. Razlika je u tome što ActionScript jezik upućuje na jednostavnost u korišcenju. U modelu sa više nivoa, Fleks aplikacija služi kao prezentacioni nivo.

Za razliku od aplikacija baziranih na HTML, Fleks aplikacije omogućavaju da za veće promene strane nije neophodno učitati novu stranu, tj. Fleks i Fleš plejer omogućavaju više načina da se podaci šalju i primaju od strane komponenti sa servera bez potrebe da klijent učita ponovo stranu. Ovakav način obrade i manipulacije sa podacima smanjuje broj potencijalnih grešaka, brži prikaz podataka i puno ugodniji korisnički interfejs. Takođe potrebno je i manje resursa.

Primer programa 2.3 ,,ZdravoSvete" napisanog u Fleksu i rezultat (slika 2.9):

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
4                 xmlns:s="library://ns.adobe.com/flex/spark"
5                 xmlns:mx="library://ns.adobe.com/flex/mx"
6                 minWidth="955" minHeight="600">
7    <fx:Script>
8      <![CDATA[
9          import spark.events.TextOperationEvent;
10         protected function obradi_ulaz(event:TextOperationEvent):
```

```
            void
11        {
12          izlaz.text = "Zdravo " + ulaz.text + "!";
13        }
14      ]]>
15  </fx:Script>
16
17  <s:TextInput id="ulaz"
18               x="5" y="20"
19               change="obradi_ulaz(event)"/>
20
21  <s:Label id="izlaz"
22           x="5" y="5"
23           text="Zdravo! "/>
24
25 </s:Application>
```




Slika 2.9: Program ZdravoSvete u Fleksu

U datom primeru možemo da vidmo da se program sastoji iz deklartivnog dela i ActionScript. Na samom početku se nalazi XML i `Aplication` deklaracija koja importuje neophodne pakete za rad. U fleksu definisan je skup komponenti `spark`. Domen imena ovih komponenti počinje sa `s` u ovom primeru `<s:Application>`. Kompnente za unos i ispis podataka takođe se nalaze u istom domenu i definisane su sa atributom kojie predstavlja identifikaciom (`id`) tog objekta, tako da im se može pristupiti i kroz ActionScript. ActionScript je odvojen `CDATA` koji označava deo XML koji se ne parsira. U tom delu je definisana funkcija koja povezuje ulaznu i izlaznu komponentu. Od datoteke MXML kompajliranjem dobijamo swf datoteku koja se umeće u html i prikazuje u veb pretraživač kroz JavaScript.

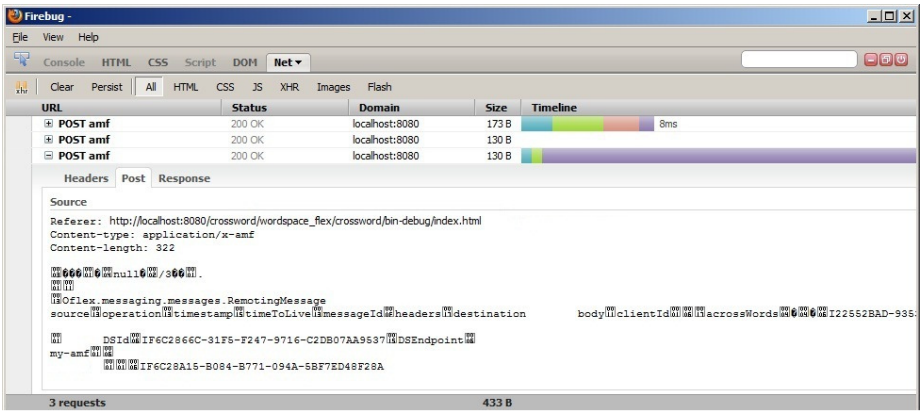
U Fleksu slanje i prijem podataka se obavlja pomoću tri klase `WebService, HTTPService i RemoteObject`. `WebService` je vezan za WSDL pomoću dokumenata (eng. Web Services Description Language) i SOAP protokolom (eng. Simple Object Access Protocol). `HTTPService` koristi HTTP metode `get, post`... U aplikaciji su iskorićeni RemoteObject-i koji za primanje i slanje poruka koriste AMF protokol (eng. Action message format) slika 2.10. Koraci za pristup Veb servisu se sastoje u eksportovanju određene javne Javine klasa u XML. U slici 2.9 je dat primer za kreiranje `RemoteObject` objekta.

```
1 <mx:RemoteObject id="RemoteLoginUser" destination="java">
2   <mx:method name="login" result="loginResultHandler(event)"/>
3   <mx:method name="register" result="registerResultHandler(event)"
        />
```

```
4 </mx:RemoteObject>
5 \label{fig:RemoteObject}
```

Fleks Bilder 4 (eng. Flex Builder 4) je okruženje, zasnovan na Eklips integrisanom razvojnom okruženju (eng. Eclipse IED), koji Adobe preporučuje kao najbolje za rad. Od kada je Fleks u potpunosti postao otvoren kod, mnogi Veb programeri su odlučili da zamene standardni način izrade Veb stranica sa novijim, poboljšanim modelom koji dozvoljava korisnicima bogatu interakciju i približava ih uobičajenim aplikacijama, ali u Veb okruženju.

**Komunikacija klijent-server** Apač Tomket - Tomket (eng. Apache Tomcat - Tomcat) je Veb server ili servlet kontejner za Javu. Ovaj čist [8] Veb server pruža podršku za Java tehnologije - servlete i JSP (eng. Java Server Pages). Najčešće preko porta 8080 se pristupa Tomket serveru.




Slika 2.10: Komunikacije klijenta i servera preko AMF objekata

Adobe BlazeDS[9] je aplikacija napisana u Javi i dostupnog je koda (eng. open source). Pokreće se unutar Jave EE i omogućava komunikaciju Jave i Fleksa. Klijentska aplikacija preko BlazeDS koristi pozive udaljenih procedura (eng. Remote procedure call - RPC) za komunikaciju sa Java aplikacijama pokrenutim na Veb serveru. Za prenos poziva i odgovora, koristi se protokol AMF protokol, preko transportnog sloja HTTP protokola. AMF je binarni protokol i njegove poruke su binarni objekti serijalizovani na klijentskoj (ActionScript objekti) odnosno serverskoj strani (Java objekti). Prednost korišćenja BlazeDS naspram drugih tehnologija (npr. SOAP, REST, WSCL itd.) je velika brzina protokola AMF postignuta binarnom serijalizacijom i malom veličinom AMF paketa i mogućnost mapiranja ActionScript i Java objekata.

## 2.4 Projektni obrasci

Projektni obrasci (eng. design pattern) su opšte, ponovo upotrebljivo rešenje za česte probleme koji se sreću prilikom projektovanja softvera. Omogućavaju da se brže i jednostvnije pravi softver koji je fleksibilniji i lakše se održava, ali

[8] Čist za Javu pisan u Javi
[9] `http://opensource.adobe.com/wiki/display/blazeds/BlazeDS`

to nije gotov dizajn koji se može direktno pretvoriti u izvorni kod. Projektni obrasci su šablon prilagođen da reši neki opštiji projektni problem u posebnom kontekstu. Projektni obrasci ne specificiranju konkretne klase i objekate.

Ideja o projektnim obrascima nastala je u knjizi *Bezvremeni načini gradnje (eng. The Timeless Way of Building)* autora Kristofer Aleksandera (nem. Christopher Alexander) u kojoj je izložio koncepte gradnje i arhitekture. ,,Svaki projektni obrazac opisuje problem koji se stalno ponavlja u našem okruženju i potom opisuje suštinu rešenja problema tako da se to rešenje može iskoristiti milion puta, a da se dva puta ne ponovi na isti način", napisao je Kristofer Aleksander. Kao nova tehnika objektno orijentisanog programiranja dobila je popularnost 1994. godine kada je ,,velika četvorka"[10] objavila knjigu *Projektni obrasci : Elementi za višestruku upotrebu u objektno orjentisanom softveru (eng. Design Patterns: Elements of Reusable Object-Oriented Software)*

Prema [ST04] projektni obrazci su grupisani u kategorije: obrasci kreiranja, obrasci strukture i obrasci ponašanja.

Obrasci kreiranja su projektni obrasci koji se bave načinom kreiranje objekata koji odgovaraju određenoj situaciji. Nekad osnovni način kreiranja objekata može da dovede do povećanja kompleksnosti dizajna a ovi obrasci rešavaju ovaj problem kontrolom kreiranje objekta. U ovu grupu se ubrajaju obrasci: apstraktna fabrika (eng. abstract factory), graditelj (eng. builder), fabrički metod (eng. factory method), lenja inicijalizacija (eng. lazy initialization), prototip (eng. prototype), skupina objekata (eng. object pool), unikat (eng. singleton)

Obrasci strukture su projektni obrasci koji olakšavaju dizajn tako što identifikuju lake načine ostvarivanja veza među različitim objektima. Primeri obrasca strukture su: adapter (eng. adapter, wrapper ), most (eng. bridge), kompozicija (eng. composite), fasada (eng. fasade), muva (eng. flyweight), iterator (eng. iterator), dekorater (eng. decorator), proksi (eng. proxy)

Obrasci ponašanja su projektni obrasci koji olakšavaju komunikaciju između objekata. Primeri uzoraka ponašanja su: posmatrač (eng. observer), iterator (eng. iterator), strategija (eng. strategy), šablonski metod (eng. template method ), stanje (eng. state), podsetnik (eng. memento), posrednik (eng. mediator), komanda (eng. command), lanac odgovornosti (eng. chain of responsibility), posetilac (eng. visitor), interpreter (eng. interpreter)

U nastavku će biti opisani projektni obrasci koji su korišćeni u implementaciji.

Projektni obrazac fabrički metod ,,definiše interfejs za kreiranje objekata, ali da ostavlja potklasama da odluče čije objekte kreiraju"[ST04]. Fabrički metod dopušta klasi da delegira stvaranje objekta potklasi. Opštije, fabrički metod se često koristi da označi bilo koju metodu čija je osnovna svrha kreiranje objekata. Primenjuje se kada treba da prenese pravilo o instanciranju objekata u neku izvedenu klasu. U tom slučaju, metoda se nalazi u objektu koji je odgovoran za to ponašanje. U nastavku se nalazi primer programa u kome je iskorišćen projektni obrazac fabrički metod.

```
1 abstract class Rec{}
2
3 class HorizontalnaRec extends Rec{}
4
5 class VertikalnaRec extends Rec{}
6
```

[10] eng. "Gang of Four" - četri autora [Wikb]

```
7  class Kreator
8  {
9    public static enum VrstaReci {horizontalna, vertikalna}
10
11   public static Rec fabrickiMetod(VrstaReci vrstaReci) {
12     switch(vrstaReci) {
13       case horizontalna:
14         return new HorizontalnaRec();
15       case vertikalna:
16         return new VertikalnaRec();
17     default:
18       return null;
19     }
20   }
21 }
22
23 public class Main {
24   public static void main(String[] args) {
25    ArrayList<Rec> reci = new ArrayList<Rec>();
26    reci.add(Kreator.fabrickiMetod(Kreator.VrstaReci.horizontalna));
27    reci.add(Kreator.fabrickiMetod(Kreator.VrstaReci.vertikalna));
28   }
29 }
```

Projektni obrazac apstraktna fabrika ,,pruža interfejs za pravljenje grupe povezanih ili zavisnih objekata bez dodavanja njihovih potpunih klasa"[ST04]. Na primer, prilikom rada sa korisničkim okruženjem, program može da koristi jedan skup objekata za rad na jednom operativnom sistemu i drugi skup objekata u drugom operativnom sistemu. Ovaj projektni obrazac omogućava da sistem uvek dobije prave objekte u određenoj situaciji. U nastavku primer prikazuje korišćenje apstraktne fabrike projektnog obrasca. Na slučajan način određuje koja se konkretna fabrika objekat kreira, da bi se zatim ona koristila kroz interfejs apstraktne fabrike. Proizvedeni konkretni proizvodi se koriste kroz interfejs apstraktnog proizvoda.

```
1  public interface ApstraktnaFabrika {
2    ApstraktniProizvod kreirajProizvod();
3  }
4
5  public interface ApstraktniProizvod {
6    public String imeProizvoda();
7  }
8
9  public class FabrikaJedan implements ApstraktnaFabrika {
10   public ApstraktniProizvod kreirajProizvod() {
11     return new ProizvodFabrikeJedan();
12   }
13 }
14
15 public class ProizvodFabrikeJedan implements ApstraktniProizvod {
16   public String imeProizvoda() {
17     return "Jedan";
18   }
19 }
20
21 public class FabrikaDva implements ApstraktnaFabrika {
22   public ApstraktniProizvod kreirajProizvod() {
23     return new ProizvodFabrikeDva();
24   }
25 }
```

```
26
27 public class ProizvodFabrikeDva implements ApstraktniProizvod {
28   public String imeProizvoda() {
29     return "Dva";
30   }
31 }
32
33 public class Main {
34   public static void main(String arg[]) throws Exception {
35     boolean random = new Random().nextBoolean();
36     ApstraktnaFabrika fabrika = null;
37
38     if (random == true)
39       fabrika = new FabrikaJedan();
40     else
41       fabrika = new FabrikaDva();
42
43     ApstraktniProizvod product = fabrika.kreirajProizvod();
44     System.out.println(product.imeProizvoda());
45
46   }
47 }
```

## 2.5 Slučajevi korišćenja

Slučaj korišćenja (eng. use case) je deo softverskog inženjeringa, odnosno oblasti koja izučava tehnike za razvoj softvera. Koristi se da precizira ponašanje sistema, bez otkrivanja njegove unutrašnje strukture.[11]

Uvešćemo nekoliko pojmova [Sta10]. Učesnik (eng. actor) predstavlja ulogu koju čovek, hardverski uređaj ili drugi sistem može da ima u odnosu na softver koji se stvara. Slučaj korišćenja je skup akcija koje vrši sistem, koje proizvode vidljiv rezultat koji je, po pravilu, od vrednosti za učesnika. [BS] Model slučajeva korišćenja se sastoji od učesnika, slučaja korišćenja i opisuje funkcionalne zahteve sistema u terminima slučajeva korišćenja.

Slučaj korišćenja se sastoji od specifikacija slučajeva upotrebe (tekstuelna reprezentacija) i dijagrama slučajeva upotrebe (vizuelna reprezentacija). Specifikaciju slučajeva upotrebe čine kratke sekvencijalno poređane rečenice u formatu naziv, svrha, akteri, učesnici, preduslov, osnovni scenario i alternativni scenario. U ovom delu nećemo ulaziti u detaljan opis specifikacija slučajeva upotrebe. Potrebno je da format bude dosledan, definisan sa jasnim rečenicama, da svaka akcija bude jasan, da ne budu u međusobnoj interakciji... Dijagram slučajeva upotrebe predstavlja se UML dijagramima. Pregledanjem modela slučajeva korišćenja lako se razume šta softver radi. Predstavlja pomoć za identifikaciju objekata u softveru, pomoć u planiranju projekta za analizu i projektovanje. Istovremeno predstavljaju odlične slučajeve testiranja, dokumetaciju za korisnika i podršku, tako da se nalazi u svim fazama razvoja softvera.

[11] Pojam je uveo Ivara Jakobsona (šve. Ivar Jacobson) 1992. godine, autora i Objedinjenog jezik za modelovanje (eng. Unified Modeling Language - UML) .

# Glava 3

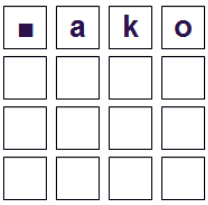
# Primena problema zadovoljavanja ograničenja na problem generisanja ukrštenih reči

Problem generisanja ukrštenih reči može se podvesti pod problem zadovoljavanja ograničenja. Promenljive su polja. Domen za svako polje je skup slova. Ograničenja predstavljaju uslovi da svaki niz horizontalnih i vertikalnih polja mora formirati validnu reč (reč koja postoji u rečniku). Ograničenja takođe može definisati korisnik tako što će zadati vrednosti za neka polja (unarna ograničenja).

U ovom radu korišten je drugi pristup, u kome su promenljive cele reči koje se popunjavaju. Domen za svaku promenljivu je skup reči iz rečnika određene dužine (dužine tražene reči iz ukrštenice). Ograničenja predstavljaju uslovi da svake dve reči koje se ukrštaju moraju imati isto slovo na poziciji na kojoj se ukrštaju. Korisnik takođe može uneti ograničenja tako što će zadati vrednosti za neka polja ili cele reči. Od opisanih tehnika u poglavlju Tehnike rešavanja implementirana je pretraga sa povratnim skokovima (data na strani 8) i pretraga sa proveravanjem unapred (data na strani 13). Promenljive tj. reči ukrštenice mogu da se biraju na tri načina: za popunjavanje bira se ona reč ukrštenice koja ima najmanji domen, zatim ona koja ima najviše popunjenih slova i slučajna reč. Vrednosti sa kojima se popunjavaju reči ukrštenice su reči iz rečnika. Reči iz rečnika mogu da se biraju na tri načina: bira se prva reč iz rečnika koja zadovoljava ograničenja, slučajna reč iz rečnika i reč sa najjačim težinskim koeficijentom (heuristika data na strani 15).

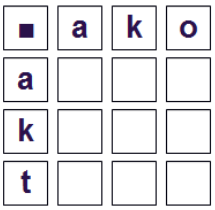
Algoritam rešavanja je podeljen na nekoliko delova koji se ponavljaju u iteracijama:
– Biranja jedne od slobodnih reči iz ukrštenice
– Biranja reči iz rečnika
– Vraćanja uz stablo pretrage, ukoliko je potrebno
Razmotrimo tok programa pri jednom generisanju ukrštenice: program učitava inicijalnu ukrštenicu koja se na početku šalje serveru od strane klijenta, prove-

rava se validnost ove ukrštenice, tj. da li sadrži nedozvoljene karaktere, popunjava se niz reči i započinje se rešavanje ukrštenice. U primeru koji će biti opisan koristićemo algoritme koji za rešenje reči u ukrštenici iz baze uzima prvu reč koju pronađe tako da se poklapa sa regularnim izrazom koji se traži. Za sledeću reč koja se rešava bira se reč sa najviše popunjenih slova. Crno polje se nalazi u prvoj vrsti i prvoj koloni ukrštenice. Izabira se reč sa najmanje praznih polja što je u našem slučaju ili prva horizontalna ili prva vertikalna reč. Izabira se prva horizontalna reč i pretražuje se u bazi reč koja se može smestiti na željenu lokaciju. U ovom konkretnom slučaju to je reč od 3 slova. Zatim se proverava da svaka vertikalna reč ima bar po jedno rešenje ukoliko se popuni izabrana horzontalna reč. Ukoliko postoji onda se reč ubacuje u ukrštenicu prilikom čega se odvijaju i dodatni procesi: stek popunjenih reči se popunjava sa popunjenom rečju, tj. u našem slučaju prvom horizontalnom, zatim se popunjavaju slova u rečima koje se ukrštaju sa popunjenom, u našem slučaju druga, treća i četvrta uspravna reč, i memorišu se lokacije slova koja su popunjavanjem ove reči dodata u ukrštenicu.



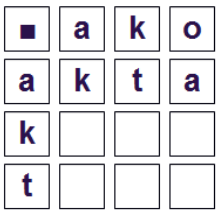

Slika 3.1: Primer kreiranja ukrštenice: prvi korak

Zatim se bira sledeća nepopunjena reč, sa najmanje nepopunjenih slova, u ovom slučaju prva uspravna reč, popunjava se rečju akt, i postupak se ponavlja.




Slika 3.2: Primer kreiranja ukrštenice: drugi korak

Bira se sledeća reč za popunjavanje, druga horizontalno. Ukoliko za ponuđenu reč iz baze ne postoji bar jedno rešenje, ona se dodaje u skup već biranih reči koje se nadalje neće birati za to mesto, bira se sledeća reč iz baze i postupak se nastavlja. U ovom slučaju za reč adut ne postoje rešenja za sve reči, pa se ona dodaje u skup probanih reči za drugu horizontalno reč, i iz baze se bira sledeća reč. Ovde je to reč akta.




Slika 3.3: Primer kreiranja ukrštenice: treći korak

Postupak se nastavlja izborom sledeće reči sa najmanje nepopunjenih polja u našem slučaju druga uspravno. Izabira se reč akti.


Slika 3.4: Primer kreiranja ukrštenice: četri korak

Za popunjavanje se izabira treća horizontalna reč. Ako u bazi ne postoji rešenje za ovu reč tako da i za ostale reči koje se ukrštaju sa njom ne postoji bar jedno rešenje u bazi, onda se vrši povratak. Povratak predstavlja brisanje skupa probanih reči i radi se vraćanje na prethodnu popunjenu reč. Prethodno popunjena reč se pronalazi pomoću steka popunjenih reči i brišu se slova koja su popunjena popunjavanjem ove reči. Pokušava se naći drugo rešenje za nju i postupak se nastavlja. U ovom slučaju ne postoji rešenje za treću horizontalnu reč tako da se može popuniti četvrta uspravna reč te se mora izvršiti povratak. Na steku se nalazi druga vertikalna reč, popunjena slova t i i se brišu i algoritam se nastavlja popunjavanjem druge uspravne reči.

Ukoliko su sva slova popunjena ukrštenica je uspešno rešena, a ako je pri pokušaju povratka stek popunjenih reči prazan, rešenje se ne može generisati i vraća se poruka o neuspešnosti.

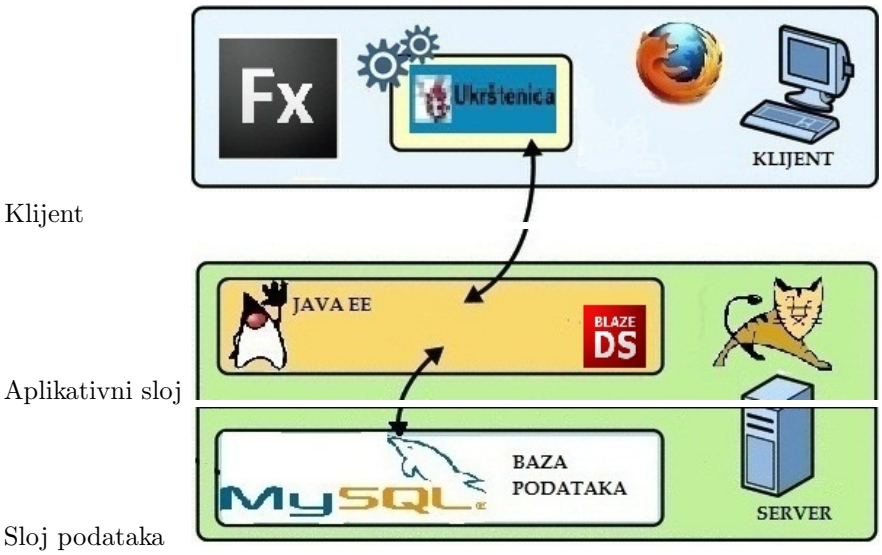
# Glava 4

# Implementacija

U ovom poglavlju biće opisana impementacija Veb aplikacije koja generiše ukrštene reči. Aplikacija je dostupna na Veb adresi `http://mavijana.dyndns.org:8080/crossword/index.html`

## 4.1 Ukrštenica

U nastavku, u nekoliko poglavlja opisana je implementacija generisanja ukrštenih reči. Ceo projekat je nazvan imenom *Crossword*. Za implementaciju Veb aplikacije korištena je Java EE 1.6.18 i baza podataka MySQL server verzija 5.1.43 za izradu serverskog dela aplikacija. Klijent strana realizovana je koristeći Adobe Fleks 4 (`RemoteObject` iskorišćeni koji za primanje i slanje poruka koriste AMF protokol). Veza između klijenta i servera realizovana je pomoću Tomketa 6.0 Veb servera, BlazeDS 3.2 (slike 4.1).




Slika 4.1: Korišćeni alati pri implementaciji

### 4.1.1 Server

U ovom poglavlju je opisana priložena implementacija serverskog dela generatora ukrštenih reči. Radi što veće fleksibilnosti i razgraničavanja odgovornosti, program se sastoji od nekoliko paketa. Ti paketi su:

**Komponente** (`components`) – klase koje čine ukrštenicu (`paket crossword.components`)

**Rečnik** (`vocabulary`) – klase za rad sa rečima (`paket crossword.vocabulary`)

**Mehanizam** (`engine`) – klase koje implementiraju algoritam za pretragu (`paket crossword.engine`)

**Server** (`server`) – klase za rad sa klijentom (`paket crossword.server`)

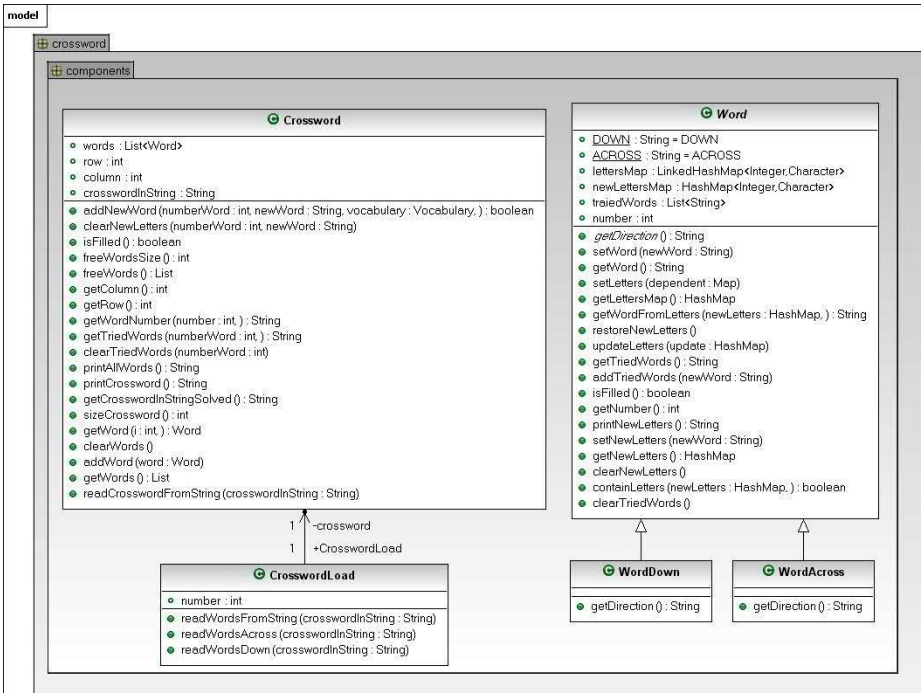
**Komponente** (`components`) U paketu `crossword.components` nalaze se klase:

- `Field`
- `Word`
- `WordAcross`
- `WordDown`
- `CrosswordLoad`
- `CrosswordGenerator`

Klasa `Field` predstavlja jedno polje ukrštenice. Sadrži statičke promenljive za prazno odnosno crno polje, `Field.EMPTY_FIELD` odnosno `Field.SQUARE_FIELD` karakter, respektivno.

Klasa `Word` predstavlja reč u ukrštenici tj. niz polja u koji se može upisati reč određene dužine. Sadrži podatake i faktički predstavlja srž čuvanja podataka ukrštenice. U sebi na početku kao inicijalno stanje ima slova koja korisnik zadaje za zadatu reč, a kasnije kako se odmiče sa rešavanje, i prazna polja u njoj se popunjavaju. Slova reči se čuvaju u `lettersMap`, kao par pozicija-slovo, tj. imaju u sebi poziciju karaktera u ukrštenici, kao i samu vrednost tog karaktera. Pošto se algoritam rešavanja ukrštenice zasniva na rešavanju jedne po jedne reči, tako i svaki objekat klase `Word` sadrži u sebi podatke koje data reč svojim popunjavanjem menja u ukrštenici. Ta slova se čuvaju u `newLettersMap`, i takođe predstavljaju par, pozicija-slovo, tj. sadrži u sebi poziciju karaktera, kao i samu njegovu vrednost. Svaki objekat klase `Word` u sebi sadrži i listu reči kojima smo pokušali da popunimo isti. Ovo nam je veoma važno za rad algoritma pretrage, jer prilikom rešavanja ukrštenice može doći do nekonzistentnosti podataka u njoj, i moramo poništiti neku reč i naći joj drugu vrednost. Lista probanih reči nam govori kojim vrednostima ne trebamo više popunjavati dati objekat i čuvaju se u triedWords. Redni broj reči kojim je svaki `Word` jednoznačno određen se nalazi unutar `number`. Još treba pomenuti da se u klasi word čuvaju i statičke vrednosti down i across koje određuju pravac reči u ukrštenici. One se koriste u klasama `WordAcross` i `WordDown`, koje nasleđuju klasu `Word` i koje detaljnije određuju reč u ukrštenici tj. njen pravac.

Klasa `CrosswordLoad` kreira inicijalno stanje ukrštenice iz stringa koji šalje korisnik i dimenzija ukrštenice. Ona kreira objekte klase `Word`, popunjava ih početnim vrednostima i dodeljuje im redni broj. U stringu koji korisnik šalje su slova iz ukrštenice nadovezana red po red, krećući se odozgo na dole tj. konkatenirani redovi ukrštenice. Odnos između klasa `Crossword`, `Word`, `WordAcross` i `WordDown` je prikazan na klasnom dijagramu (slika 4.2).



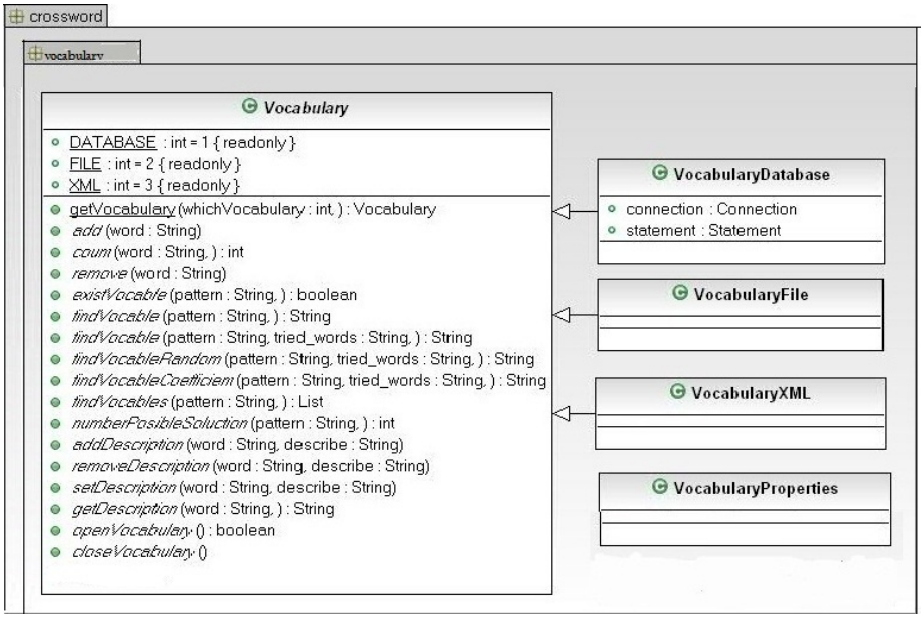

Slika 4.2: Klasni dijagram paketa `components`

**Rečnik** (`vocabulary`) Za performanse rešavanja ukrštenice veoma bitnu ulogu ima komunikacija sa rečnikom. Pri svakom punjenju ukrštenice, pri svakom odabiru reči, svaki put kada primenimo algoritam pretrage, mi moramo da se obratimo bazi, i najveći deo vremena se provodi u metodama ove klase. U paketu `crossword.vocabulary` nalaze se sledeće klase:

- `Vocabulary`
- `VocabularyDatabase`
- `VocabularyFile`
- `VocabularyXML`
- `VocabularyProperties`

`Vocabulary` je apstraktna klasa koja definiše ponašanje ostatka sistema prema rečniku i obrnuto. Može se koristiti nekoliko različitih izvora perzistentnih podataka (datoteku, relaciona baza...) i u isto vreme zadržati uniforman interfejs rečnika prema ostatku sistema. Realizovan prema DAO projektnom uzorku zajedno sa projektnim uzorcima apstraktna fabrika i fabrički metod tako da `Vocabulary` obezbeđuje metod `getVocabulary(whitchVocabulary)` koji konstruiše određenu instancu rečnika. U ovoj klasi se nalaze i statičke promenljive `Vocabulary.DATABASE`, `Vocabulary.FILE` i `Vocabulary.XML` koje se unose kao argument getVocabulary(). Ako je potrebno dodati novi tip rečnik, sistem ne trpi velike izmene. Potrebno je dodati novu klasu koja je nasleđuje `Vocabulary`, implementirati metode i u klasu `Vocabulary` dodati novi tip rečnika. Na ovaj način je implementirana klasa `VocabularyDatabase`. Ostale klase nisu još implementirane. U klasi `PropertiesVocabulary` je definisano nekoliko statičkih objekata koji služe za podešavanje sistema (u ovom slučaju parametri MySQL

baze podataka). Odnos između ovih klasa je prikazan na klasnom dijagramu (slika 4.3).




Slika 4.3: Klasni dijagram paketa `vocabulary`

**Mehanizam** `(engine)` Klase koje implementiraju ovaj algoritam smeštene su u paketu `crossword.engine`. Te klase su:

- `CrosswordGenerator`
- `StateKeeperStack`
- `FindVocable`
- `FindVocableFirst`
- `FindVocableWithCoefficient`
- `NextWord`
- `NextWordDVOMRV`
- `NextWordLeastFree`

`CrosswordGenerator` ima samo jednu statičku metodu:

```
generate(
Crossword crossword,
StateKeeperStack stateKeeperStack,
Vocabulary vocabulary,
NextWord nextWord,
FindVocable findVocable),
```

koja definiše osnovu algoritma. Rezultat je status o uspehu, odnosno neuspehu kreiranja ukrštenice.

Algoritam rešavanja je podeljen na nekoliko delova koji se ponavljaju u iteracijama

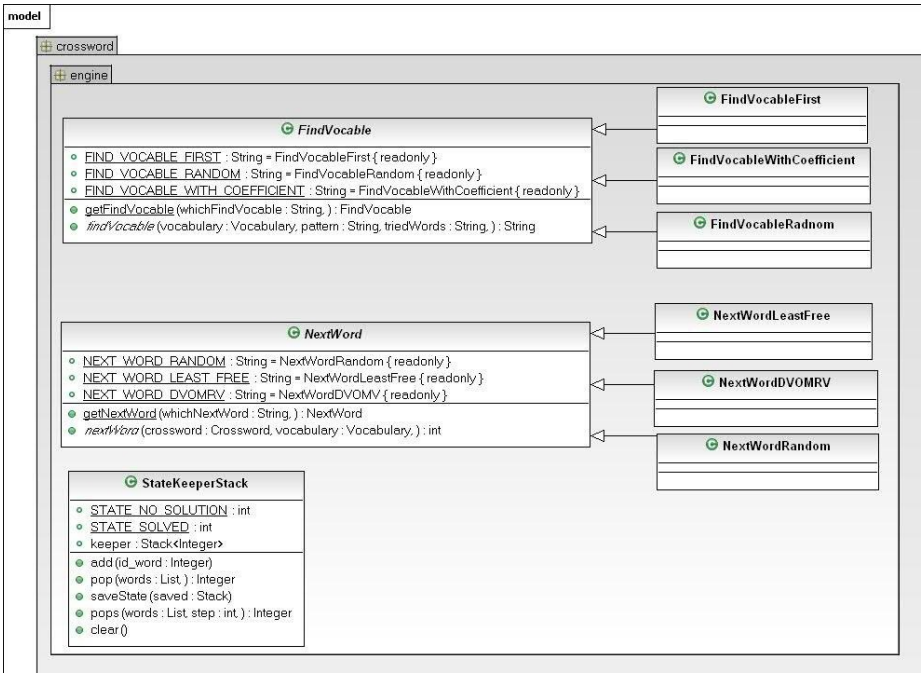
– Biranja jedne od slobodnih reči iz ukrštenice

– Biranja reči iz rečnika

– Vraćanja uz stablo pretrage, ukoliko je potrebno

U klasi `StateKeeperStack` je implementiran stek za čuvanje id popunjenih reči. Kada se na steku nalazi reč iz ukrštenice za koju nema rešanja sklanja se data reč i newLetters i triedWord se čiste. Elementarne operacije `NextWord` i `FindVocable` su izdvojene u zasebne abstraktne klasa tako se različitim implementacijama ovih klasa mogu, bez menjanja ostatka sistema, implementirati različite strategije u rešavanju problema.

Apstraktna klasa `NextWord` ima apstraktnu metodu `nextWord` koja vraća id nepopunjene reči iz ukrštenice. U slučaju da su sve reči iz ukrštenice popunjene vraća konstantu koja označava da je popunjena ukrštenica. U slučaju da za prvu izabranu reč u ukrštenici ne postoji reč iz baze, vraća se konstanta koja označava da je ukrštenica nerešiva. `NextWord` je implementiran u sledećim klasama: `NextWordRandom` koji bira slučajnu reč iz skupa reči koje odgovaraju paternu. Zatim `NextWordLeastFree` koji bira za popunjavanje reč u kojoj ima najmanje nepopunjenih slova. `NextWordDVOMRV` koji bira reč za popunjavanje koja u bazi ima najmanje mogućih rešanja. Odgovarajuća implemantacija `NextWord` klase može da se dobije pozivom statičke metode `getNextWord(whichNextWord)` sa odgovarajućom konstantom koje su definisane u klasi `NextWord`. Apstraktna klasa `FindVocable` takođe definiše apstraktnu metodu `findVocable` koja vraća traženu reč iz baze ili vraća taj patern u slučaju nepronalaska reči. `FindVocable` je implementiran u tri varijante: `FindVocableFirst` vraća prvu pronađenu reč iz rečnika; `FindVocableRadnom` vraća na slučajan izbor reči;
`FindVocableWithCoefficient` vraća reč koja ima prioritet prema težinskim koeficijentima za svako slovo. Odnos između ovih klasa je prikazan na klasnom dijagramu (slika 4.4).



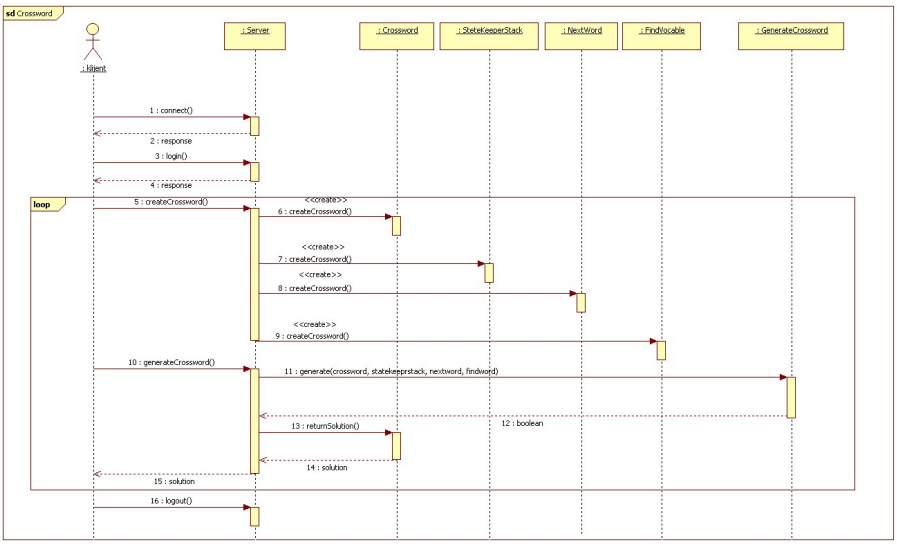

Slika 4.4: Klasni dijagram paketa `engine`

**Server** (`components`) U paketu `crossword.server` nalaze se sledeće klase:

- `LoginUser`
- `Server`
- `Properties`
- `PropertiesLogin`

Klasa `LoginUser` je zadužena za početno logovanje korisnika i za kreiranje novog naloga. Klasa `Server` definiše funkicije za rad sa klijentom. Funkcije za kreiranje i generisanje ukrštenice. Omogućava podešavanje algoritma za rešavanje. Podrazumevani algoritmi se nalaze u klasi `Properties` definisani kao statičke promenljive ove klase.

Kada se pokrene Veb pretraživač, prvo se šalje zahtev za konekciju na server. Na taj zahtev odgovara klasa `Server` metodom `connect()`, Nakon uspešnog logovanja klijent može da podesi željene paremtre za ukrštenicu i započne generisanje ukrštenice. Server odgovara sa ukrštenicom ili upozorenjem da nije moguće generisati takvu ukrštenicu i daje mogućnost za dalje pokušaje. Na slici 4.5 prikazan je sekvancijalni dijagram komunikacije klijenta i servera ali bez neuspešnog scenarija.




Slika 4.5: Sekvancijalni dijagram komunikacije servera i klijenta

**Poređenje algoritama** Pri generisanju ukrštenice četri kolone i četri vrste, pri algoritmu pretrage sa povratnim skokovima dobijaju se približni rezultati koji su prikazani u tabeli 4.1 a za pretragu sa proveravanjem unapred rezultati su u tabeli 4.2 rečnik je implementiran preko MySQL baze. U tebelama 4.3 i 4.4 rezultati dobijeni na isti način a rečnik je implementiran preko fajla[1].

[1]Rezultati su dobijeni kao srednja vrednost više pokretanja aplikacije

| rečnik: MySQL baza | reč ukrštenice koja ima najmanji domen | reč koja ima najviše popunjenih slova | slučajna reč |
|---|---|---|---|
| bira se prva reč iz rečnika | 4s | 5s | 1s |
| slučajna reč | 1s | 3s | 7s |
| reč sa najjačim koeficijentom | 4s | 0.2s | 33s |

Tabela 4.1: Rezultati rešavanja pri izboru pretraga sa povratnim skokovima

| rečnik: MySQL baza | reč ukrštenice koja ima najmanji domen | reč koja ima najviše popunjenih slova | slučajna reč |
|---|---|---|---|
| bira se prva reč iz rečnika | 4s | 5s | 1s |
| slučajna reč | 1s | 3s | 82s |
| reč sa najjačim koeficijentom | 0.8s | 14s | 230s |

Tabela 4.2: Rezultati rešavanja pri izboru pretrage sa proveravanjem unapred

| rečnik: datoteka | reč ukrštenice koja ima najmanji domen | reč koja ima najviše popunjenih slova | slučajna reč |
|---|---|---|---|
| bira se prva reč iz rečnika | 3.4s | 0.5s | 3.7s |
| slučajna reč | 0.5s | 5.1s | 1.3s |
| reč sa najjačim koeficijentom | 4.3s | 6.9s | 1.3s |

Tabela 4.3: Rezultati rešavanja pri izboru pretraga sa povratnim skokovima

| rečnik: datoteka | reč ukrštenice koja ima najmanji domen | reč koja ima najviše popunjenih slova | slučajna reč |
|---|---|---|---|
| bira se prva reč iz rečnika | 1.5s | 0.2s | 1.3s |
| slučajna reč | 2.3s | 1.1s | 0.9s |
| reč sa najjačim koeficijentom | 3.5s | 2.8s | 1.8s |

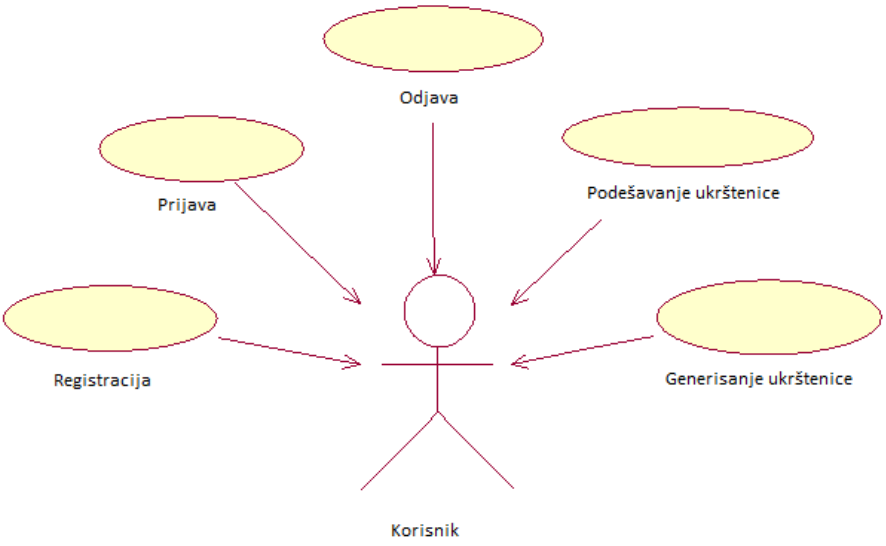
Tabela 4.4: Rezultati rešavanja pri izboru pretrage sa proveravanjem unapred

### 4.1.2 Klijent

U ovom poglavlju je opisana organizacija klijentskog dela generatora. Sastoji se iz tri paketa:
– Komponente `(components)` - grafički elementi `(paket crossword.components)`
– Podatke `(data)` - padatke koje čini slike, pomoćne xml datoteke, pomoćne grafičke komponente `(paket crossword.data)`
– Ukrstenica `(crossword )` - sadrži već pomenute elemente ujedinjene i klasu Generate_functions za komunikaciju sa serverom.

Detaljnije implementacija klijenta biće prikazana kroz slučajeve korišćenja.

**Verbalan opis** Korišćenje klijenta je sastavljeno iz par koraka. Nakon logovanja na početnoj stranici, korisnik se nalazi na stranici na kojoj može da podesi kakvu ukrštenicu želi da kreira: veličinu, algoritam za rešavanje, crna polja i da unese željena slova. Nakon podešavanja i akcije servera, dobija se ukrštenica sa objašnjenjima za svaku reč. Definišemo slučajeve korišćenja (slika 4.6):
– slučaj korišćenja – Registracija
– slučaj korišćenja – Prijava
– slučaj korišćenja – Odjava
– slučaj korišćenja – Podešavanje ukrštenice
– slučaj korišćenja – Generisanje ukrštenice




Slika 4.6: Diagram slučajeva korišćenja klijenta

**Registracija**

**SK1: Slučaj korišćenja – Registracija**
**Naziv**: Registracija
**Svrha**: Služi za registraciju novih korisnika na sistem
**Akteri**: Korisnik

**Učesnici**: Korisnik i sistem
**Preduslov**: Korisnik nije prijavljen na sistem

**Osnovni scenario**:
1. Korisnik bira opciju Registracija.
2. Korisnik unosi potrebne podatke i potvrđuje ih.
3. Sistem proverava unete podatke i snima podatke o novom korisniku u bazu.

**Alternativni scenario**:
A.1 Ukoliko korisnik nije uneo u formu neki od potrebnih podataka, sistem ga obaveštava o propustu.
A.2 Ukoliko je uneo ime koje je već rezervisano, sistem ga obaveštava
A.3 Ukoliko sistem ne može da se poveže na bazu prikazuje poruku o datom problemu.

#### Prijava

**SK2: Slučaj korišćenja – Prijava** (slika 4.7)
**Naziv**: Login
**Svrha**: Služi za logovanje registrovanog korisnika na sistem
**Akteri**: Korisnik
**Učesnici**: Korisnik i sistem
**Preduslov**: Korisnik je registrovan

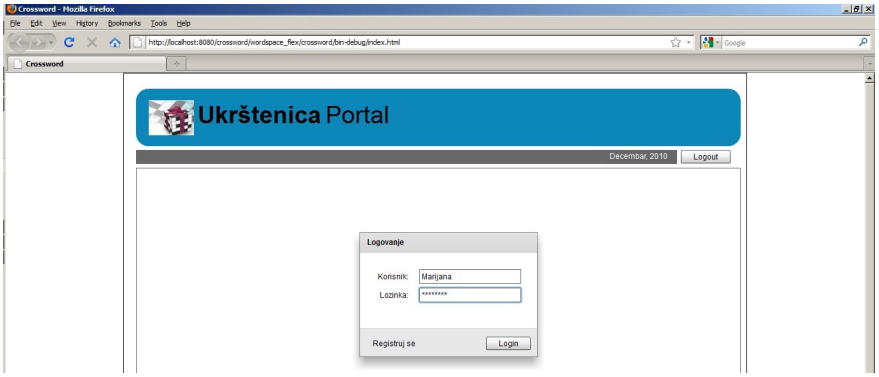
**Osnovni scenario**:
1. Korisnik da bi pristupio mora izvršiti prijavu.
2. Sistem prikazuje formu za unos korisničkog imena i lozinke
3. Korisnik unosi korisničko ime i lozinku i potvrđuje unos.
4. Sistem proverava date podatke u bazi, prijavljuje korisnika i vraća potvrdu o prijavi.

**Alternativni scenario**:
A.1 Ukoliko sistem ne može da se poveže na bazu prikazuje poruku o datom problemu.
A.2 Ukoliko u bazi ne postoji registrovani korisnik sa datim korisničkim imenom i lozinkom sistem obaveštava korisnika o tome.

#### Odjava

**SK3: Slučaj korišćenja – Odjava**
**Naziv**: Logout
**Svrha**: Služi za odjavu korisnika sa sistema
**Akteri**: Korisnik
**Učesnici**: Korisnik i sistem
**Preduslov**: Korisnik je prijavljen na sistem

**Osnovni scenario**:
1. Korisnik bira opciju Logout.
2. Sistem odjavljuje korisnika.




Slika 4.7: Prijava korisnika

### Podešavanje ukrštenice

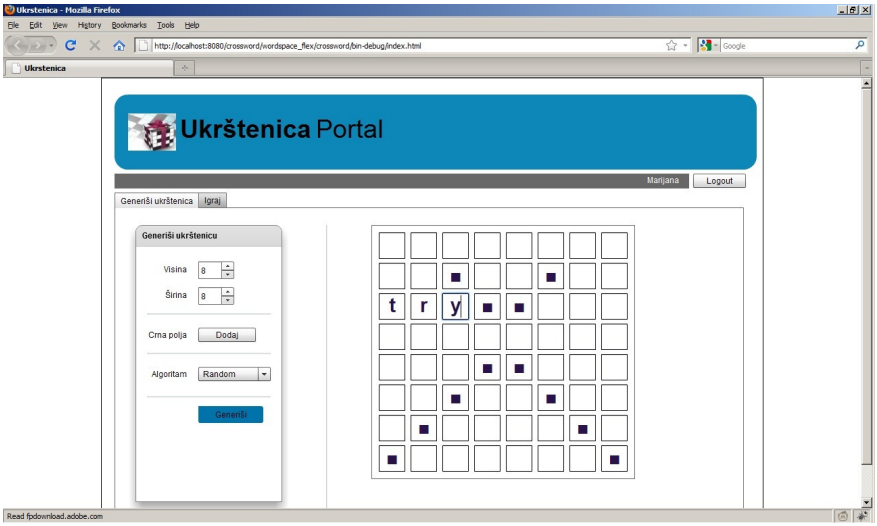
**SK4: Slučaj korišćenja – Podešavanje ukrštenice** (slika 4.8)
**Naziv**: Visina, Širina, Dodaj, Izaberi
**Svrha**: Služi za podešavanje ulazne ukrštenice za koju je potrebo izgenerisati rešenje
**Akteri**: Korisnik
**Učesnici**: Korisnik i sistem (bez servera)
**Preduslov**: Korisnik je prijavljen na sistem

**Osnovni scenario**:
1. Korisnik podešava visinu i širinu
2. Dodavanje crnih polja započinje klikom na dugme 'Dodaj'. Svaki klik na neko polje ukrštenice pretvara se u crno polje, a klik na crno polje briše crno polje. Završetak dodavanja crnih polja izvodi se ponovnim pritiskom na isto dugme. Izlaskom iz režima dodavanje crnih polja nemamo mogućnost brisanja postojećih polja osim ponovnim vraćanjem na ovu akciju.
3. Izbor alogritma za rešavanje se svodi na odabir algoritma iz liste.
4. U svakom trenutku moguće je dodati slovo na željenu poziciju u ukrštenici osim kada je uključena opcija za dodavanje crnih polja.

**Alternativni scenario**:
A.1.Ograničena veličina ukrštenice

### Generisanje ukrštenice

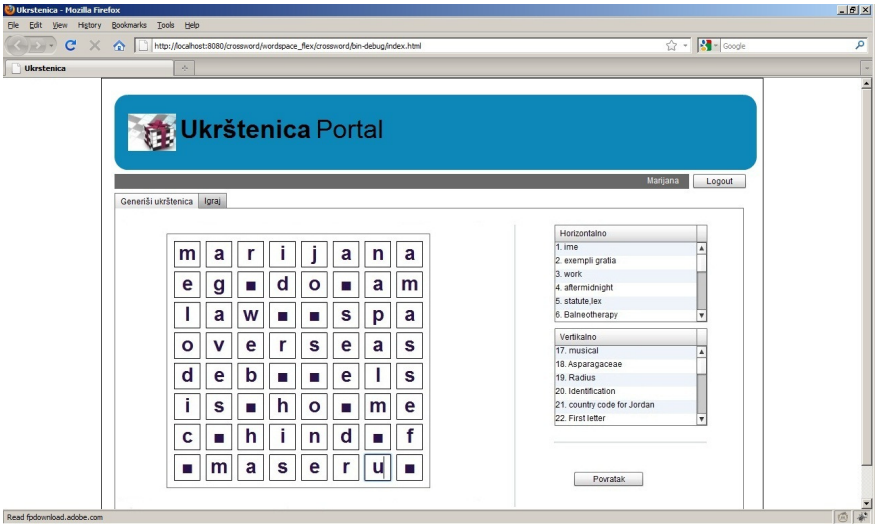
**SK5: Slučaj korišćenja - Generisanje ukrštenice** (slika 4.9)
**Naziv**: Generiši
**Svrha**: Korisnik poziva sistem da reši postavljenu ukrštenicu
**Akteri**: Korisnik
**Učesnici**: Korisnik i sistem
**Preduslov**: Korisnik je prijavljen na sistem




Slika 4.8: Podešavanje ukrštenice

**Osnovni scenario**:
1. Korisnik klikom na dugme 'Generiši' traži od sisteme da genriše ukrštenicu
2. Biranje opicije za 'Povratak' korisnik se vraća na SK4

**Alternativni scenario**:
A.1. Ukoliko ne postoji rešenje ukrištenice sistem obaveštava korisnika




Slika 4.9: Generisana ukrštenica

# Glava 5

# Zaključak

Ovaj rad prevashodno se bavi opisivanjem problema zadovoljavanja ograničenja, i njihovom primenom na konkretnom problemu. Kroz date primere se može videti da je jednostavnije rešavati probleme prilagođavajući ih postojećim i već rešenim problemima, nego rešavajući ih nezavisno. Vremenom se tehnike razvijaju i unapređuju, a skup generalizovanih problema je sve veći, tako da je rešavanje problema korišćenjem problema zadovoljavanja ograničenja sve jednostavnije i efikasnije.

U aplikaciji za generisanje ukrštenih reči upotrebljeno je više alata i oni se mogu besplatno preuzeti sa Interneta (Java, MySQL, Fleks kao i neophodne biblioteke). Logika se obavlja na serverskoj strani tako da je moguće klijentsku stranu uraditi u nekom drugom programskom jeziku, kao i uslužiti veći broj klijenata.

Rešenje je kompletno. Omogućava korisniku kreiranje novih ukrštenica, izbor veličine i rasporeda crnih polja. Korisnik može zadati neka od polja ili čitave reči. Sistem podržava unicode UTF-8 kodni raspored tako da je jednostavnom zamenom rečnika moguće kreirati ukrštenice na različitim jezicima. Delovi algoritma su definisani interfejsima tako da je promena algoritma tj. implementacija još nekih optimizacionih tehnika moguća različitim implementacijama ovih interfejsa. Od mogućih unapređenja mogu se pomenuti implementacija više izvora rečnika, dodavanje opcije za čuvanje ukrštenice, unapređenje efikasnosti algoritma pretrage, implementaciju još nekih tehnika provere konzistentnosti pri izboru reči ili drugačije heuristike za izbor mesta vraćanja pri konfliktu, dodavanje mogućnosti igranje ukrštenica i javna dostupnost.

# Bibliografija


[And00] Gregory R Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison Wesley, 2000.

[BF71] J. N. G. Brittan and F. J. M. Farley. College timetable construction by computer. *The Computer Journal vol.14*, pages 361–365, 1971.

[BS] Kurt Bittner and Ian Spence. *Use case modeling.*

[BW02] Chen X. van Beek P. Bacchus, F. and T. Walsh. *Binary vs. non-binary constraints.* 2002.

[Cha] Philip Chan. Constraint satisfaction problems. `http://www.cs.sfu.ca/~mori/courses/cmpt310/slides/chapter05.pdf`. [Online; accessed 19 July 2011].

[Clo71] M. B. Clowes. *Artijicial Intelligence.* 1971.

[Coo71] S. A. Cook. *The Complexity of Theorem-Proving Procedures.* STOC, 3rd edition, 1971.

[HE80] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, pages 263–313, 1980.

[Huf71] D. A. Huffman. *Impossible objects as nonsense sentences.* 1971.

[IG10] Antoaneta Klobučar Iva Gregurić. Problem četiri boje. (10):21–29, 2010.

[JM94] M. D. Johnston and G. E. Miller. Spike: Intelligent scheduling of hubble space telescope observations. *In Intelligent Scheduling.*, pages 391–422, 1994.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms.* Addison Wesley, 1996.

[Mac92] Alan K. Mackworth. The logic of constraint satisfaction. *Artif. Intell.*, pages 3–20, 1992.

[Mar09] Filip Marić. Formalizacija, implementacija i primene, sat rešavača. 2009.

[MF90] Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *AAAI'90*, pages 25–32, 1990.

[MS98] Kim Marriot and Peter J. Stuckey. *Programming with constraints: An introduction*. MIT Press, 1998.

[RN03] Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[Sch95] R. Schulte. Three-tier computing architectures and beyond. 1995.

[SH80] Stephen D. Shapiro and Robert M. Haralick. Editorial. *Pattern Recognition*, pages –1–1, 1980.

[ST04] Alan Shalloway and James R. Trott. *Projektni obrasci*. Mikro knjiga, 2004.

[Sta10] Staša Vujičić Stanković. Informacioni sistemi. 2010.

[Tsa93] Edward P. K. Tsang. Foundations of constraint satisfaction. Computation in cognitive science, 1993.

[Vla09] Siniša Vlajić. Projektovanje softvera. 2009.

[Wal75] D Waltz. *Understanding line drawings of scenes with shadows*. 1975.

[Wika] Wikipeida. Distribuirano računarstvo. `http://bs.wikipedia.org/wiki/Distribuirano_ra%C4%8Dunarstvo`. [Online; accessed 19 July 2011].

[Wikb] Wikipeida. Gang of four. `http://en.wikipedia.org/wiki/Object-oriented_programming#Gang_of_Four_design_patterns`. [Online; accessed 27 July 2011].

# Terminološki rečnik

## Srpsko-engleski

| | |
|---|---|
| algoritam minimalnih konflikta | min-conflicts algorithm |
| aplikacioni server | application server |
| aplikacioni sloj | application tier |
| aplikacioni programski interfejs | Application programming interface - API |
| binarni problem zadovoljavanja ograničenja | binary constraint satisfaction problems |
| distribuirani | distributed |
| dvostruko konzistentan | 2-consistency |
| Java enterprajz izdanje | Java Enterprise Edition - Java EE |
| jednostruko konzistentan | 1-consistency |
| $k$-konzistentan | $k$-consistency |
| konzistentnost čvorova | node consistency - NC |
| konzistentnost luka | arc consistency - AC |
| konzistentnost puta | path consistency - PC |
| markiranje | backmarking - BM |
| povratni skokovi | backjumping |
| pretraga sa povratkom | backtracking |
| prezentacioni sloj | presentation tier |
| problem zadovoljavanja ograničenja | constraint satisfaction problems - CSP |
| promenljivu sa najmanjim domenom | minimum remaining value - MRV |
| sloj podataka | data tier |
| Veb server | Web server |

## Englesko-srpski

| | |
|---|---|
| 1-consistency | jednostruko konzistentan |
| 2-consistency | dvostruko konzistentan |
| $k$-consistency | $k$-konzistentan |
| arc consistency - AC | konzistentnost luka |
| application programming interface - API | aplikacioni programski interfejs |
| application server | aplikacioni server |
| application tier | aplikacioni sloj |
| backjumping | povratni skokovi |
| backmarking - BM | markiranje |
| backtracking | pretraga sa povratkom |
| binary constraint satisfaction problems | binarni problem zadovoljavanja ograničenja |
| constraint satisfaction problems - CSP | problem zadovoljavanja ograničenja |
| data tier | sloj podataka |
| distributed | distribuirani |
| Java Enterprise Edition - Java EE | Java enterprajz izdanje |
| min-conflicts algorithm | algoritam minimalnih konflikta |
| minimum remaining value - MRV | promenljivu sa najmanjim domenom |
| node consistency - NC) | konzistentnost čvorova |
| path consistency - PC | konzistentnost puta |
| presentation tier | prezentacioni sloj |
| Web server | Veb server |