UNIVERZITET U BEOGRADU
SAOBRAĆAJNI FAKULTET
SMER ZA POŠTANSKI SAOBRAĆAJ I MREŽE

ZAVRŠNI RAD

OBJEKTNO ORIJENTISANI SIMULACIONI MODEL ŠALTERSKE SLUŽBE U POŠTI 11158 BEOGRAD

Mentor:
Prof. dr Milorad Stanojević,dipl.inž.

Student:
Njegoš Slavković

Beograd, 2014. godina

Rezime

U radu je prezentovan objektno orijentisani simulacioni model realizovan u opštem programskom jeziku C++ i baziran na simulacionom modelu kreiranom u simulacionom jeziku GPSS/FON. Uporednom analizom rezultata oba simulaciona modela prikazane su sličnosti i razlike, kao i prednosti i mane, ova dva pristupa rešavanja istog problema.

Rešavanje problema počinje izlaganjem opštih karakteristika sistema masovnog opsluživanja, statističkim utvrđivanjem raspodela verovatnoća dolazaka i vremena opsluga na šalterima, kreiranjem simulacionog modela i prikazivanjem osnovnih parametara koji karakterišu funkcionisanje poštanskih šaltera.

Detaljno je objašnjen objektno orijentisani pristup, u prilogu je prezentovan model u C++ jeziku kao i sve dobijene statistike.

Abstract:

The paper presented an object oriented simulation, coded in general purpose programming language C++, based on simulation model created in simulation language GPSS/FON. Comparing results from both models we wanted to show similarities and differences between the two as well as pros as cons of those two different approaches to the same problem.

Solving problems begins with presentation of the general characteristics on queuing systems, statistical determination of the probability distribution od arrival times and passenger handling at the counters, the creation of simulation models and displaying basic parametars that characterize the functioning of postal counter.

We have explained objected oriented aproach in details, presented the whole C++ model in appendix with all general statistics.

SADRŽAJ

Pregled slika:

Strane

Pregled tabela:

Pregled algoritama:

UVOD

Efikasno funkcionisanje šalterskih službi u jedinicama poštanske mreže je jedan od osnovnih preduslova za kvalitetno pružanje poštanskih usluga. Šalterske službe su tačke u kojima se odvijaju prva i poslednja faza pružanja poštanskih usluga, pa se u njima vrši interakcija i neposredna komunikacija između korisnika i preduzeća poštanskog saobraćaja. Imajući u vidu da je jedna od osnovnih funkcija pošte zadovoljenje potreba korisnika za komuniciranjem, upravo je šalterska mreža deo poštanskog sistema kome treba posvetiti dosta pažnje prilikom organizovanja i održavanja.

Šalterske službe u jedinicama poštanske mreže se mogu posmatrati kao sistemi masovnog opsluživanja sa čekanjem i neograničenim brojem mesta u redu. Analiza kvaliteta njihovog rada zasniva se na primeni teorije masovnog opsluživanja, statističke analize, računarske simulacije uz poznavanje postojećeg načina njihovog funkcionisanja. U teoriji se kvalitet opsluživanja utvrđuje na osnovu tri kriterijuma, od kojih se kao apsolutni kriterijum usvaja princip da srednje vreme koje klijenti provedu u redu mora biti manje od srednjeg vremena koje provedu na opsluzi.

Cilj svakog donosioca odluke, prilikom planiranja ili poboljšavanja funkcionisanja ovih sistema, jeste da se sa jedne strane teži da iskorišćenost opslužioca bude što veća, a sa druge strane da korisnici budu zadovoljni u pogledu vremena zadržavanja u sistemu i redovima čekanja. Pronalaženje optimalnog rešenja zahteva da se odluka unapred proveri, da se ispita stabilnost rešenja, sprovedu dopunski eksperimenti i uvedu dodatne informacije, pa da se na osnovu toga odabere rešenje i uz što manji rizik donese odluka. U pojedinim fazama istraživačkog projekta izvođenje eksperimenta na realnim sistemima nije moguće zbog njihove kompleksnosti, dugotrajnosti procesa, velikih novčanih usluga ili bezbednosti istraživanja, pa je najbolje rešenje formiranje modela na osnovu realnih sistema na kojima će se vršiti sva ispitivanja i analiziranja. Za relativno jednostavne sisteme parametri sistema se mogu dobiti analitički, ali za analizu sistema opsluživanja u poštama najčešće se koristi digitalna simulacija.

U poređenju sa drugim metodama analize, digitalna simulacija ima niz pogodnosti kao što su fleksibilnost, jednostavnost, univerzalnost, ekonomičnost, brzina , dr. i u znatnoj meri omogućava prevazilaženje problema složenosti koji je jedna od osnovnih karakteristika velikog broja realnih sistema. Ovaj metod se može koristiti u fazi projektovanja pre izgradnje sistema, a i kao sredstvo analize u cilju predviđanja promena kod postojećeg sistema. Proces simulacije podrazumeva dve osnovne faze: fazu izgradnje modela i fazu eksperimentisanja nad modelom uz analiziranje dobijenih rezultata. Model je osnovna pretpostavka simulacije i on podrazumeva pojednostavljenu sliku realnog sistema. Prilikom njegovog kreiranja moraju se odabrati karakteristike realnog sistema koje su značajne za dobijanje potrebnih parametara, a da se pri tom vodi računa da model ne bude suviše složen, ni suviše jednostavan. Složeni modeli daju rezultate veoma slične realnim, ali su po pravilu preskupi i neadekvatni za eksperimentisanje, dok sa druge strane previše pojednostavljeni modeli ne odslikavaju u potpunosti realnim sistem, a dobijeni rezultati mogu da budu neadekvatni i pogrešni. Simulacija podrazumeva analiziranje konkretnog sistema, određivanje nivoa detaljnosti modela, granice između sistema i okoline i eliminisanje nepotrebnih ili manje bitnih elemenata i veza. Dalje se opisuje sistem sa svim njegovim komponentama, načinom rada i povezanosti sa okolinom. Za formiranje modela neophodni su tačni i kvalitetni ulazni podaci o realnom sistemu.

U ovom radu će biti ispitana efikasnost funkcionisanja šalterske službe u pošti 11158 Beograd, da bi se utvrdilo da li je kvalitet opsluživanja korisnika zadovoljavajuća. Šalterska služba ove pošte je sistem masovnog opsluživanja koji će biti posmatran i analiziran. Na osnovu podataka koji su dobijeni merenjem pristupa korisnika u sistem i vremena potrebnih za njihovo opsluživanje biće formirani simulacioni model.

1. Sistemi masovnog opsluživanja

Sistemi masovnog opsluživanja postoje u različitim delatnostima u kojima se javlja potreba za opsluživanjem većeg broja ljudi ili mašina. Među najjednostavnije primere spada opsluživanje kupaca u radnjama ili prodaja karata za pozorište, bioskop i prevoz, sistemi za opravku i održavanje opreme, obezbeđivanje razgovora preko telefona, ukazivanje medicinske pomoći u ambulanti, utovar i istovar brodova u luci, sletanje i uzletanje aviona na aerodromu i dr. Poštanska šalterska služba predstavlja tipičan sistem masovnog opsluživanja.

U svakom sistemu koji je naveden postoje tri suštinski bitna elementa: ulazni tok – način na koji klijenti dolaze, disciplina opsluživanja – način na koji klijenti čekaju na opslugu i mehanizam opsluživanja – način na koji će klijenti biti opsluženi. Proces opsluživanja može da se okarakteriše najrazličitijim pokazateljima: brojem sredstava za opsluživanje i brojem ljudi u organizaciji koja vrši opsluživanje, brzinom opsluživanja, dužinom reda, vremenom čekanja na opsluživanje, vremenom opsluživanja, brojem otkaza od opsluživanja i dr.[1]

Kod ovih sistema se često javlja problem da organizacija koja opslužuje raspolaže ograničenim brojem kanala, pa ne mogu istovremeno biti opsluženi svi klijenti. Neravnomernost u pojavljivanju klijenata na mesto opsluživanja dovodi do nakupljanja zahteva za opsluživanjem i do formiranja redova čekanja. Problem smanjenja redova čekanja i mnogi drugi organizacioni problemi sistema masovnog opsluživanja uspešno se rešavaju matematičkim metodama (analitički i numerički), pomoću računarske simulacije koja će u radu detaljno biti opisana. Nezavisno od načina rešavanja, ovi modeli omogućavaju analitičarima da precizno procene parametre sistema.

1.1.Teorija masovnog opsluživanja

Teorija masovnog opsluživanja je deo teorije verovatnoće koja koristi njen matematički alat za rešavanje mnogih praktičnih zadataka koji se javljaju u sistemima masovnog opsluživanja. Ključni element tih sistema su klijenti i opslužioci. Pod klijentom se podrazumeva proizvoljan zahtev za opsluživanjem. U navedenim primerima klijenti su ljudi koji kupuju robu u radnjama, putnici koji žele da kupe kartu, pacijenti koji u bolnici čekaju pomoć lekara, brod koji ulazi u luku itd. Klijenti u sistem pristupaju u proizvoljnim, slučajni trenucima iz konačne ili beskonačne populacije. Kod sistema sa velikom populacijom klijenata u pitanju je beskonačna populacija. Tipičan primer beskonačne populacije su korisnici poštanskih usluga. Glavna razlika između konačne i beskonačne populacije je u tome kako je definisan intezitet korisnika. Kod modela sa beskonačnom populacijom intezitet korisnika je nezavisan od broja klijenata koji su napustili populaciju potencijalnih klijenata i pristupili sistemu masovnog opsluživanja. U modelima sa konačnom populacijom intezitet dolazaka zavisi od broja klijenata koji su bili opsluženi ili čekaju. Proces dolazaka u modelima sa beskonačnom populacijom se obično karakteriše vremenom između dolazaka sukcesivnih klijenata.

Dolasci se mogu dešavati u raspoređenim ili slučajni vremenima. Kada se dolasci slučajni, vremena između dolazaka su obično okarakterisana raspodelom verovatnoće[1].

Skup svih klijenata koji pristupaju sistemu naziva se *ulazni tok korisnika.* Mnogobrojni proračuni koji su izvedeni prilikom rešavanja različitih zadataka teorije masovnog opsluživanja pokazali su da se u većini slučajeva može dobiti zadovoljavajuće rešenje ako se pretpostavi da su tokovi klijenata Puasonovi[1].

Ukoliko ne mogu biti opsluženi istog trenutka kada pristupe sistemu, klijenti staju u *red čekanja*. Sredstva kojima se vrši opsluživanje klijenata nazivaju se *kanalima opsluživanja.* To su šalteri u pošti ili banci, biletarnice, piste za uzletanje i sletanje aviona, kase u radnjama itd. U tabeli 1.1 mogu se videti primeri sistema masovnog opsluživanja sa svim njihovim osnovnim delovima[1].

Tabela 1.1 *primeri sistema masovnog opsluživanja*

SISTEM	TOK KLIJENATA	REDOVI ČEKANJA	NAČIN OPSLUŽIVANJA	KANALI OPSLUŽIVANJA
Pošta, banka	Klijenti koji zahtevaju uslugu	Čekanje da šalter bude slobodan	Pružanje poštanskih i bankarskih usluga	Šalteri
Saobraćaj	Putnici	Čekanje na nailazaka prevoznog sredstva	Prevoz putnika i robe	Autobusi, vozovi automobili
Aerodromi	Avioni	Avioni koji kruže iznad aerodroma	Sletanje i uzletanje aviona	Avionske piste
Telefonske centrale	Klijenti koji podižu slušalicu	Čekanje na signal Telefonske centrale	Prenos poruka	Komunikacioni sistemi

Kao što se u prethodnoj tabeli vidi, osnovni delovi sistema masovnog opsluživanja su ulazni tok klijenata i kanali opsluživanja. Pod analizom ovih sistema podrazumeva se analiziranje svih parametara sistema i ustanovljenje veze između broja kanala opsluživanja i broja klijenata. Jasno je da se veći broj klijenata može opslužiti ukoliko postoji veći broj kanala opsluživanja, ali je isto tako očigledno da je veliki broj kanala opsluživanja povezan sa velikim rashodima. Zadatak analize jeste pronalaženje optimalnog rešenja koje će zadovoljiti i zahteve klijenat ali i održati poslovanje rentabilnim, u zavisnosti od toga do koje granice može ići preduzeće ka ispunjenju želja klijenata[1].

Sistemi masovnog opsluživanja se razlikuju prema ulaznim tokovima klijenata, prema raspodelama verovatnoća vremena opsluživanja i prema disciplini opsluživanja.[1]

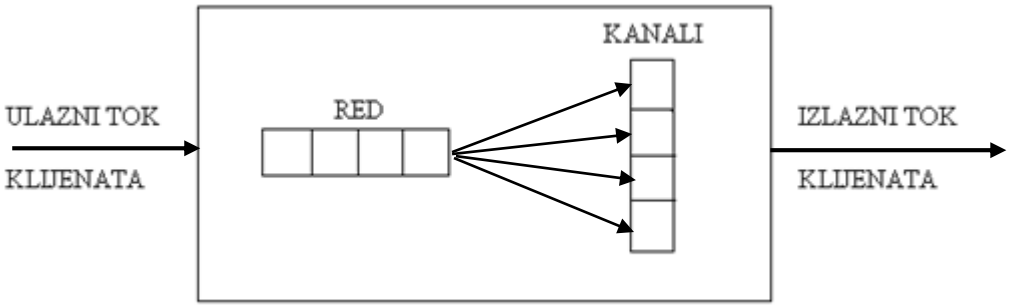
Disciplina opsluživanja klijenata ukazuje na redosled klijenata u redu i određuje koji će klijent i kada biti opslužen. Klijenti se mogu opasluživati prema sledećim kriterijumima[2]:

- FIFO(First In, First Out) – klijent koji je prvi ušao u sistem, prvi će biti opslužen;
- LIFO(Last In, Lasr Out) – klijent koji je poslednji ušao u sistem, prvi će biti opslužen;
- SIRO(Service In Random Order) – metod slučajnog izbora, pojavljuje se kod telefonskih sistema;
- PR(PRiority) – u ovim sistemima postoje prioritetni klijenti koji će biti opsluženi pre ostalih;
- SRT(Shortest Remainig Time) – disciplina po kojoj se opsluživanje vrši u zavisnosti od vremena.

Osnovni tipovi sistema masovnog opsluživanja jesu[1]:

- Sistemi sa čekanjem klijenata u redu
- Sistemi sa otkazima klijenata od opsluživanja

Kada klijent pristupi sistemu masovnog opsluživanja sa čekanjem on može odmah biti opslužen ako postoji slobodan kanal opsluživanja. Ukoliko su svi kanali zauzeti on staje u red i čeka sve dok se neki od njih ne oslobodi. U sistemima sa neograničenim brojem mesta u redu klijent može dobiti otkaz. Međutim, ukoliko su vreme čekanja ili broj mesta u redu ograničeni, klijenti u nekom trenutku mogu dobiti otkaz. Na slici 1.1 je grafički prikazan način funkcionisanja sistema masovnog opsluživanja sa čekanjem[1].



Slika 1.1 *Funkcionisanje sistema masovnog opsluživanja sa čekanjem*

Za sistem opsluživanja kažemo da je sa otkazima ako se klijent opslužuje odmah po dolasku u sistem, ili ako ga napušta a nije opslužen (dobija otkaz) u slučaju kada zatekne sve kanale zauzete. Kao primer ovakvih sistema se može navesti automatska telefonska centrala: ako je tražena telefonska linija već zauzeta, tj. ako ne postoji ni jedan slobodan kanal, klijent dobija otkaz. Međutim, ako naiđe u momentu kada je slobodan makar jedan od njih, biće primljene na opsluživanje i do kraja se opslužiti [1].

1.2.Parametri sistema masovnog opsluživanja sa čekanjem

Da bi se Teorija masovnog opsluživanja primenila na neki sistem, moraju biti definisani njegovi ulazni paramtri odnosno njegove karakteristike, a tu se podrazumevaju:

- broj kanala opsluživanja koji se obeležavaju sa *n*. U sistemu koji se ovde ispituje to je broj šaltera koji vrše opslugu korisnika;
- intezitet ulaznog toka korisnika koji se obeležava sa λ. To je broj korisnika koji pristupe sistemu u toku jednog minuta;
- intezitet opsluživanja korisnika μ, odnosno broj opsluženih korisnika u minuti

$$\mu=\frac{1}{t} \qquad (1)$$

Ovo su parametri na osnovu kojih sistem može biti u potpunosti određen. Međutim, postoje i neki koji bliže određuju njegove karakteristike i biće navedeni u nastavku.

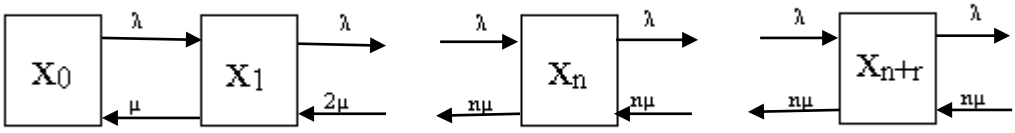
- $\bar{t}$ je srednje vreme opsluživanja korisnika
- ρ je intezitet saobraćaja koji se u sistemu izvrši, odnosno srednji broj korisnika koji pristupaju u sistemu masovnog opsluživanja za srednje vreme opsluživanja jednog korisnika

$$\rho = \lambda \cdot \bar{t} \qquad (2)$$

- α je stepen zauzetosti kanala i srednji broj korisnika koji prispevaju u sistem masovnog opsluživanja za srednje vreme opsluživanja jednog korisnika pomoću svih kanala

$$\alpha = \frac{\lambda}{n\mu} = \frac{\rho}{n} \qquad (3)$$

Na slici 1.2 je prikazana graf stanja sistem masovnog opsluživanja sa čekanjem sa intezitetima prelaza stanja iz jednog u drugo. Vidimo da sistem može imati više stanja. Kada se nalazi u stanju x_0 znači da nema klijenata koji žele opslugu. Stanje x_k podrazumeva to da se u sistemu nalazi k klijenata i svi se opslužuju, a u stanje x_{n+r} nalazi se n klijenata koji se opslužuju i r klijenata koji čekaju u redu za opsluživanje(r=1,2,3….m). Sistem prelazi iz stanja u x_0 u stanje x_1 sa intezitetom λ i sa istim u svako susedno stanje.



Slika 1.2 graf stanja sistema masovnog opsluživanja sa čekanjem

Na osnovu datog grafa izvode se i izračunavanja verovatnoća stanja sistema. U radu neće biti prikazano izvođenje, već samo konačne formule. U stacionarnom režimu rada gde je λ = *const* i μ = *const* , postoje sledeće verovatnoće stanja:

- verovatnoća da će sistem biti slobodan

$$p_0 = \frac{1}{\sum_{i=0}^{n-1}\frac{\rho^i}{i!}+\frac{\rho^n}{(n-1)!(n-\rho)}} \quad (4)$$

- verovatnoća da se u sistemu nalazi k klijenata

$$p_k = \frac{\rho^k}{k!} \cdot p_o \quad (5)$$

Osnovne karakteristike sistema masovnog opsluživanja sa neograničenim brojem mesta u redu za čekanje su:

- verovatnoća opsluživanja korisnika

$$p_{ops} = 1 \quad (6)$$

- verovatnoća otkaza

$$p_{otk} = 0 \quad (7)$$

- gubitak saobraćaja

$$G = 0 \quad (8)$$

- verovatnoća postojanja reda

$$p_{pr} = \frac{\alpha}{1-\alpha} p_n \quad (9)$$

- verovatnoća čekanja u redu

$$\pi_n = \frac{1}{1-\alpha} p_n \quad (10)$$

- srednji broj zauzetih kanala

$$\bar{n}_z = \rho \quad (11)$$

- srednji broj korisnika u redu (srednja dužina reda

$$\bar{d}_r = \frac{\alpha}{(1-\alpha)^2} p_n \quad (12)$$

- srednje vreme čekanja u redu

$$\bar{t}_r = \frac{\bar{d}_r}{\lambda} \quad (13)$$

- srednje vreme koji korisnik provede u sistemu

$$T_s = \bar{t}_r + \bar{t} \quad (14)$$

- stepen zauzetosti sistema

$$S = \frac{\rho}{n} \cdot 100\ \% \quad (15)$$

- gubitak sistema

$$\Gamma = 100\ \% - S \quad (16)$$

Specijalan slučaj sistema sa čekanjem je kada postoji samo jedan kanal opsluživanja. To je takozvani jednokanalni sistem masovnog opsluživanja sa čekanjem i neograničenim brojem mesta u redu. Verovatnoća stanja za ovakav sistem su:

- verovatnoća da u sistemu nema klijenata

$$p_n = 1- \rho \tag{17}$$

- verovatnoća da se u sistemu nalazi jedan klijent

$$p_1 = \rho \cdot (1- \rho) \tag{18}$$

U tom slučaju važi $\alpha = \rho = \frac{\lambda}{\mu}$ jer je n = 1.

Ostale karakteristike ovih sistema su:

- verovatnoća postojanja reda

$$p_{pr} = 1- p_0 - p_1 \tag{19}$$

- srednji broj korisnika u redu (srednja dužina reda)

$$\bar{d}_r = \frac{\rho^2}{1-\rho} \tag{20}$$

- srednje vreme čekanja u redu

$$\bar{t}_r = \frac{\bar{d}_r}{\lambda} \tag{21}$$

- srednje vreme koje korisnik provede u sistemu

$$T_s = \bar{t}_r + \bar{t} \tag{22}$$

- stepen zauzetosti sistema

$$S = \bar{n}_z \cdot 100\ \% \tag{23}$$

- gubitak sistema

$$\Gamma = p_0 \cdot 100\ \% \tag{24}$$

1.3.Kvalitet opsluge u sistemima masovnog opsluživanja

Dobra organizacija rada poštanske šalterske službe, kao sistema masovnog opsluživanja, s jedne strane zahteva dovoljnu iskorišćenost, odnosno, zauzetost sistema, a sa druge strane zadovoljavajući nivo opsluge korisnika. Ne treba po svaku cenu povećati kavalitet opsluge smanjivanjem redova čekanja, ako se sa druge strane ne iskorišćavaju dovoljno resursi sistema i trpe veliki gubici. Optimalno rešenje je pronalaženje sredine između ovih zahteva u zavisnosti od poslovne politike preduzeća[1].

Nivo opsluge, odnosno kvalitet opsluživanja korisnika, meri se dužinom čekanja u redu. Sistem masovnog opsluživanja se čekanjem i neograničenim brojem mesta u redu za čekanje u potpunosti je određen brojem kanala opsluživanja, intezitetom ulaznog toka i srednjim vremenom opsluživanja (n,λ i $\bar{t}$). Svakoj od ove tri veličine se može upravljati na odgovarajući način.

Broj kanala opsluživanja se može povećati i smanjivati prema potrebi, ukoliko su raspoloživi određeni kapaciteti. Pristupom korisnika u sistem se najteže može upravljati. Jedina mogućnost je usmeravanje određene kategorije na opsluživanje u vreme manjeg inteziteta saobraćaja ili zakazivanjem opsluge u vreme koje je povoljno za rad sistema.
Srednjim vremenom opsluživanja se može upravljati boljom organizacijom opsluživanja, gde se podrazumeva automatizacija delova sistema opsluge ili odlaganje poslova koji nisu neophodni u datom trenutku za vreme kada je kanal za opsluživanje slobodan[1].
U literaturi koja se bavi pitanjima kvaliteta opsluživanja u sistemima sa čekanjem, kao i u operacionim istraživanjima, za ocenu kvaliteta opsluživanja koriste se sledeće kriterijumi[1]:

- Kriterijum koji se izražava verovatnoćom čekanja korisnika u redu. Opsluživanje korisnika je kvalitetno ako je verovatnoća čekanja u redu manja od usvojene vrednosti ε (za $\varepsilon \leq 1$).

$$P_{PR} < \varepsilon \tag{25}$$

- Kao kriterijum za ocenu kvaliteta opsluživanja moguće je takođe, uzeti uslov da je srednja dužina reda čekanja manja od kapaciteta sistema, što znači da će u ovakvom sistemu biti manje korisnika u redu od broja kanala opsluživanja.

$$\bar{d}_{\mathrm{r}} < n \tag{26}$$

- kada su sistemi masovnog opsluživanja poštanske šalterske službe, uglavnom se usvaja apsolutni kriterijum za ocenu kvaliteta opsluživanja. Uslov za kvalitetnu opslugu je taj da je srednje vreme čekanja u redu manje od srednjeg vremena opsluživanja jednog korisnika.

$$\bar{t}_{\mathrm{r}} < \bar{t}_{\mathrm{ops}} \tag{27}$$

1.4.Poštanska šalterska služba kao sistem masovnog opsluživanja

Sistem službe za pružanje usluga korisnicima koji se nalaze u jedinicama poštanske mreže mogu se posmatrati kao sistem masovnog opsluživanja sa čekanjem i neograničenim brojem mesta u redu. Opsluga korisnika se vrši po FIFO principu, što znači da će prvi korisnik koji prispe u sistem prvi biti i opslužen. Klijenti koji pristupaju šalterima posmatraju se kao događaji, a njihov skup je ulazni tok događaja. Taj ulazni tok ima svoju empirijsku raspodelu koja se dobija prikupljanjem i analiziranjem statističkih podataka dobijenih na osnovu merenja frekvencije dolazaka korisnika i vremena njihove opsluge u konkretnom sistemu za koji se vrši analiza. Posle toga se može vršiti testiranje i analiziranje empirijske raspodele primenom raznim matematičkih metoda i statističkih testova kako bi se potvrdilo poklapanje sa teorijskim raspodelama na osnovu kojih se dalje mogu vršiti ispitivanja[11].

U današnjim tržišnim uslovima kada se Pošti „nameće“ prelaza s monopolističke na tržišno orijentisanu organizaciju, mora se mnogo više voditi računa o zadovoljenju potreba sadašnjih i budućih korisnika. Do sada je proizvod, odnosno, poštanska usluga bila u centru pažnje i težilo se da se prodaje u što većem obimu.

To je bilo lako ostvarivo zato što u poštanskom sistemu nije postojala konkurencija, JP PTT saobraćaja “Srbije“, današnje Javno Preduzeće „Pošta Srbije“, je imala eksluzivno pravo na pružanje poštanskih usluga i korisnici su bili „prinuđeni“ da koriste njihove usluge bez obzira da li su njima zadovoljni. Danas kada tržište poštanskih usluga postaje liberalizovano, što znači da će bilo koje preduzeće moći da pruža poštanske usluge, javlja se problem za nacionalnog operatora kako da zadrži korisnike čije se želje i zahtevi sve više povećavaju. Rešenje leži u ispitivanju zahteva i promeni načina funkcionisanja i pružanja usluga korisnicima da se povećao kvalitet servisa a ujedno i procenat zadovoljnih korisnika[11].

Redovi čekanja korisnika na opsluživanje su osnovni problem koji se javlja prilikom pružanja usluga u jedinicama poštanske mreže, a i samog poslovanja poštanskog preduzeća. Na prvi pogled, jedan od očiglednih načina za smanjenje redova može biti povećanje broja šaltera na kojima se opsluga. Međutim, uzimajući u obzir da dolasci klijenata u sistemu nisu unifirmno raspodeljeni, da se u različitim periodima rada javlja različita frekvencija broja dolazaka, nije pravo rešenje dimenzionisati kapacitete na osnovu njihovog broja u času vršnog opterećenja. Tada bi došlo do velikih gubitaka u sistemu i kapaciteti bi većim delom dana bili predimenzionisani[11].

Rešenje za problem redova čekanja dobija se primenom teorije masovnog opsluživanja na konkretne šalterske službe, simulacijama ili drugim metodama pogodnim za analiziranje sistema masovnog opsluživanja[2].

U ovom radu će sistem biti analiziran primenom objektno orjentisane simulacije u programskom jeziku C++.

2. MODELIRANJE I SIMULACIJA

2.1.Modeliranje i modeli

Pod modeliranjem se podrazumeva stvaranje uprošćenih i idealizovanih slika realnosti radi njenog boljeg i lakšeg razumevanja. Ovaj proces se javlja i ponavlja u svakodnevnim ljudskim aktivnostima i izražava sposobnost korišćenja simbola i jezika pri međusobnoj komunikaciji, sposobnost mišljenja, zamišljanja, procenjivanja i predviđanja, korišćenja iskustva odnosno uočavanje obrazaca u ponašanju i mnoge druge aktivnosti ljudskog uma. Pojednostavljene slike realnosti koje se javljaju kao rezultat modeliranja nazivaju se modelima. Uz pomoć njih se omogućava suočavanje sa realnim svetom na pojednostavljen način isključujući svu njegovu kompleksnost kao i sve opasnosti koje se mogu javiti kao posledice eksperimentisanja nad njim. Postoje i slučajevi u kojima nije moguće eksperimentisati na realnim sistemima, pa je kreiranje modela od velikog značaja za njihovo razumevanje i analiziranje. Navodimo neke od slučajeva[3]:

- Nemoguće je eksperimentisati na sistemima koji ne postoje, pre ili prilikom planiranja njihove izgradnje ili rekonstrukcije;
- Menjanje načina funkcionisanja velikih sistema za potrebe eksperimenata koji možda neće dati dobre rezultate je veoma skupo i neisplativo;
- U realnom sistemu nije jednostavno menjati parametre da bi se utvrdio način njegovog funkcionisanja pod različitim opterećenjem, a na štetu ljudi koji su deo tog sistema ili njegovi korisnici.

Model predstavlja apstrakciju realnog sistema i ima samo one karakteristike originala koje su bitne za svrhu njegovog izučavanja. To znači da se pri procesu modeliranja mora izvršiti odabir između onih karakteristika i elemenata sistema koji će biti značajni za istraživanje i uključeni u model, od onih koji u realnom sistemu postoje, ali nisu značajni prilikom istraživanja i neće biti uključeni u model jer bi ga bez potrebe činili složenim i teškim za razumevanje. Model mora biti toliko detaljan koliko je to potrebno za donošenje ispravnih odluka i zaključaka o realnom sistemu. Cilj modela nije da precizno reprodukuje stvarnost u svoj njegovoj složenosti, već da uobliči na vidljiv, često formalan način ono što je suštinski za razumevanje njegove strukture ili ponašanja[3].

Suviše složeni ili savršeni modeli koji imaju sposobnost da za isti skup ulaznih veličina proizvode iste izlazne vrednosti kao i realni sistemi, čak iako su ostvarivi, po pravilu su preskupi i neadekvatni za eksperimentisanje. S druge strane, suviše pojednostavljeni modeli ne odslikavaju na pravi način posmatrani sistem, a rezultati koji se dobijaju njihovom primenom mogu da budu neadekvatni i pogrešni[3].

Stoga je potrebno u određenom trenutku povući granicu u realnom sistemu i to tako da rezultujući model što vernije odslikava posmatrani sistem, ali i da s druge strane, njegova složenost i cena ne budu ograničavajući faktori[3].

Realni sistem se može predstaviti formalnim i neformalnim modelima. Neformalni opis modela sadrži osnovne pojmove o modelu i to, objekte – delove iz koga je izrađen model, opisne promenljive – koje opisuju stanja u kojima se objekti nalaze u određenim vremenskim trenucima, i pravila interakcije objekata koja definišu kako objekti modela utiču jedan na drugi u cilju promene njihovog stanja.

Neformalni opis modela se priprema veoma lako i brzo, ali on najčešće nije jasan, naročito kada se opisuju složeni modeli. Prilikom formiranja ovih modela može doći do grešaka koje kasnije prouzrokuju sledeće anomalije:

- Nekompletan opis modela – ukoliko model ne sadrži sve situacije koje mogu da nastupe;
- Nekonzistentan opis modela – ukoliko su za istu situaciju predviđena dva ili više pravila čijom se primenom dobijaju kontradiktorne akcije;
- Nejasan opis modela – ako u jednoj situaciji treba obaviti dve ili više akcija, a da pri tome nije definisan njihov redosled;

Da bi se izbegle ove anomalije i usavršili modeli, u savremenoj metodologiji modeliranja utvrđeni su formalizmi koji određuju klasu posmatranih objekata na nedvosmislen i generalan način korišćenja konvencija i pravila. Pomoću tih formalizama stvaraju se formalni opisi modela koji obezbeđuju veću potpunost i preciznost u opisivanju modela, a ponekad omogućuju i da se formalizuje postupak ispitivanja nekompletnosti, nekonzistentnosti i nejasnosti. Uvođenje i korišćenje formalizama omogućava potpuno usmeravanje pažnje na karakteristike objekata koje su najznačajnije za istraživanje[3].

Za izradu modela ne postoje striktna pravila kojih se treba pridržavati, ali postoje opšte preporuke koje mogu pomoći da modeli budu što verniji i korisniji[3].

- Modeli ne smeju previše složeni niti detaljni. Treba da sadrže samo relevantne elemente sistema – suviše složene i detaljne modele gotovo nije moguće vrednovati ni razumeti;
- Model ne sme suviše da pojednostavi problem;
- Granica sistema sa okolinom mora biti odabrana tako da sistem, odnosno njegov model, obuhvata samo fenomene od interesa. Okolina sistema modelira se tako da se ne opisuju detalji fenomena i uzročna veza među njima, već se daje samo njihov sažet prikaz;
- Razumno je rastaviti model na više dobro definisanih i jednostavnih modula sa tačno određenom funkcijom, koje je lakše izgraditi i proveriti;
- U razvoju modela preporučuje se korišćenje neke od proverenih metoda za razvoj algoritama i programa;
- Potrebna je provera logičke i kvantitativne ispravnosti modela, i to kako pojedinačnih modula tako i celog modela. Kod modela koji uključuju slučajne promenljive to znači i primenu odgovarajućih statističkih tehnika.

2.1.1. Vrste modela

Postoji mnogo vrsta modela koji se mogu koristiti za predstavljanje realnih sistema, kao i veliki broj načina na koji se oni mogu podeliti. U nastavku će biti predstavljeni najbitniji i najzastupljeniji modeli, odnosno neki od načina za njihovu klasifikaciju[3].

- **Mentalni modeli** su strukture koje ljudski mozak neprekidno konstruiše kako bi bio u stanju da poveže niz činjenica sa kojima se čovek susreće, a potom na osnovu toga da deluje. Takvi modeli omogućuju, na primer, razumevanje fizičkog sveta, komunikaciju među ljudima i planiranje akcija.

- **Verbalni modeli** su direktna posledica mentalnih modela i predstavljaju njihov izraz u govornom jeziku, a uobičajeno se predstavljaju u pisanom obliku. Oni spadaju u klasu neformalnih modela.
- **Konceptualni modeli** se stvaraju na osnovu predstave o strukturi i logici rada sistema ili problema koji se modelira. Oni predstavljaju osnovu za izradu računarskih modela. Često se nazivaju i strukturni, pošto u grafičkom obliku ukazuju na strukturu posmatranog sistema.
- **Matematički modeli** su oni kod kojih su veze između objekata modela opisane matematičkim (numeričkim) relacijama. Kod formulisanja matematičkog modela polazi se od verbalnog modela koji se transformisanjem dovodi u stanje koje se može opisati matematičkim jezikom. Ova klasa modela ima široku primenu, naročito u nauci i inženjerskim disciplinama.
- **Računarski (simulacioni) modeli** su prikaz konceptualnih modela u obliku programa za računar. U tom obliku modeli postaju sredstvo kojim se može efikasno analizirati rad modela u različitim spoljnim uslovima i sa različitim unutrašnjim parametrima i tako dobiti uvid u ponašanje sistema koji model opisuje. Kao sredstvo za izražavanje, računarski modeli koriste programske jezike i stoga su blisko vezani za razvoj računarskih nauka.

Drugi način za klasifikaciju modela jeste podela na [4]:

- **Fizičke modele** koji predstavljaju stvarne sisteme ili kopiju sistema malih dimenzija;
- **Šematske modele** koji sadrže mape, grafikone i dijagrame. Njima se slikovito predstavlja sistem;
- **Heurističke modele** koji podrazumevaju skup deskriptora i pravila odlučivanja koji je zasnovan na računaru i nije ograničen fizičkim, dijagramskim ili matematičkim ograničenjima. Mogu biti programirani za ispitivanje skupova podataka kao i za izvršavanje logičkih poređenja;
- **Simboličke modele** od kojih su najopštiji matematički i simulacioni modeli. Šematski i simbolički modeli igraju glavnu ulogu u planiranju izvršavanja računarske simulacije sistema. Matematički modeli koriste simboličku notaciju i matematičke jednačine za predstavljanje sistema. Simulacioni model je poseban tip matematičkog modela kod kojeg su atributi sistema predstavljeni pomoću matematičkih funkcija koje povezuju promenljive.

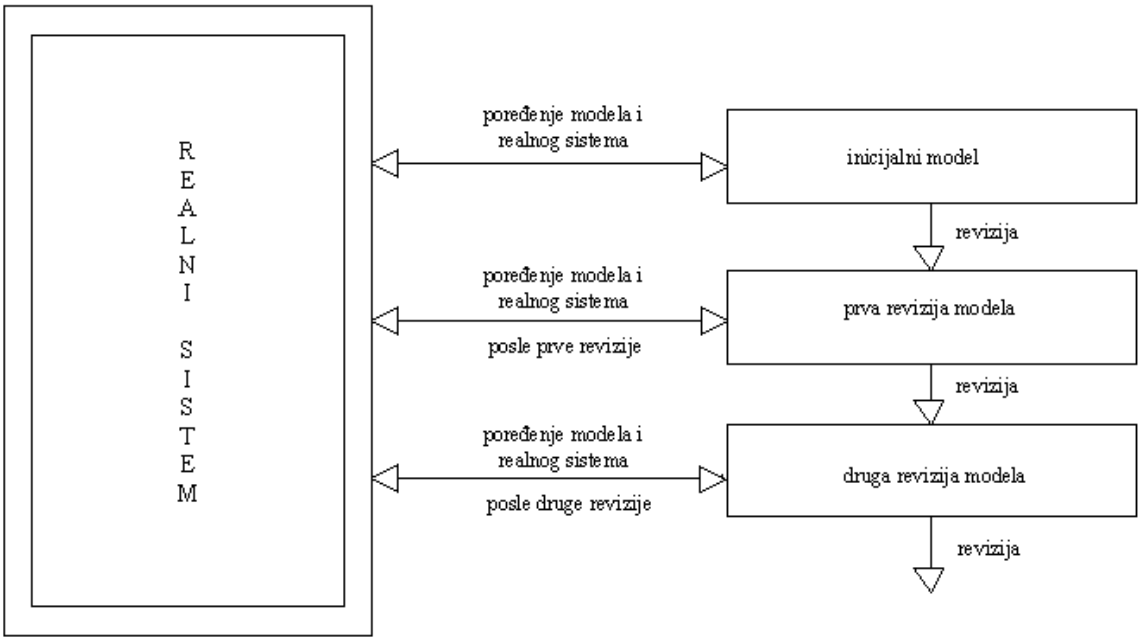
2.1.2. Validacija i verifikacija

Činjenica je da je model uvek uprošćena slika realnog sistema odnosno pojednostavljeni pogled na realni sistem koji se istražuje. Sve pretpostavke i aproksimacije koje se unose u model mogu dovesti u pitanje validnost tog modela. Zbog toga se na model uvek gleda sa određenom rezervom iz razloga što se veoma lako mogu izostaviti bitni elementi zbog koga on kasnije neće verno prezentovati željeni sistem. Iz svega toga proističe potreba validacije modela odnosno njegovog testiranja da bi se ustanovilo da li je on pouzdan, bez grešaka i da li je dovoljno uverljiv za sve one koji će ga koristiti. Validacija podrazumeva proceduru u kojoj se ponašanje modela poredi sa ponašanjem realnog sistema, a uočena neslaganja i razlike se koriste za ispravku modela.

Postupak poboljšanja modela se nastavlja i ponavlja sve dok se ne odluči da je dobijena tačnost modela zadovoljavajuća. Cilj procesa validacije modela je dvojak[3]:

- Da proizvede model koji predstavlja ponašanje realnog sistema i koji je dovoljno blizak realnom sistemu, tako da se može koristiti za obavljanje ekperimenata;
- Da pouzdanost modela poveća na prihvatljiv nivo tako da ga mogu koristiti razni donosioci odluka;

Na slici 2.1 je prikazan postupak poređenja modela i realnog sistema prilikom validacije. Ukoliko se prilikom poređenja utvrdi da između njega i modela postoje neslaganja i razlike za koje se smatra da nisu opravdane, vrši se ispravka na modelu. Zatim se tako dobijeni model ponovo poredi sa realnim sistemom i vrše nove ispravke, ukoliko je to potrebno. Poređenje modela sa realnim sistemom se vrši pomoću više različitih testova – subjektivnih ili objektivnih. Subjektivni testovi podrazumevaju učešće ljudi koji dobro poznaju sistem ili neke njegove aspekte i koji primenjujući posebna znanja o sistemu donose zaključke o modelu i njegovom izlazu. Objektivni testovi uvek zahtevaju neke podatke o ponašanju sistema, kao i podatke koje prozvodi model. Prikupljeni podaci se obrađuju različitim statističkim testovima koji porede određene apekte skupa podataka sistema i skupa podataka modela[3].



Slika 2.1 Validacija modela

Celokupan proces validacije zahteva određeno vreme, trud i novčana sredstva. Ovi faktori se svakako moraju uzeti u obzir u postupku validacije i donošenju odluke o tome koliko će se daleko otići u procesu revizije modela. Najčešće, modelar određuje maksimalni prihvatljivi nivo odstupanja modela od realnog sistema. Ukoliko se ovaj nivo odstupanja ne može postići uz ulaganje ograničenih budžetskih sredstava namenjenih procesu validacije, modelar ili smanjuje očekivanu preciznost modela ili odbacuje dati model[3].

Verifikacija predstavlja proveru da li je simulacioni program bez grešaka i konzistentan sa modelom. Iako se validacija i verifikacija modela konceptualno razlikuju, najčešće se simultano sprovode. Razlog je to što se u praksi ova dva procesa velikim delom poklapaju.

Simulacija omogućuje da se za kratko vreme izvrši kontrola promene rezultata u zavisnosti od promene ulaznih parametara, što omogućuje testiranje osetljivosti programa na varijacije samo jednog ulaznog parametra, dok ostali ostaju nepromenjeni. Testiranje na poremećaje podrazumeva unošenje neprirodnih vrednosti ulaznih parametara i praćenje ponašanja izlaznih parametara, čime se mogu otkriti greške u programu koje se veoma teško uočavaju[3].

2.2.Računarska simulacija

Simulacija je veoma mlada naučna disciplina koja je vezana za razvoj informacionih tehnologija i naučne tehnike. Može se definisati kao[1]:

- Imitacija operacija realnog, postojećeg procesa ili sistema tokom vremena;
- Serija jednačina koje su uvršćene u računarski program kojima se opisuju značajni aspekti sistema;
- Numerička tehnika za odvijanje eksperimenata koristeći određene tipove matematičkih i logičkih modela kojima se opisuje ponašanje poslovnog ili ekonomskog sistema u datom vremenskom periodu, a posredstvom računara.

Čak i pored velikog broja modela koji su izgrađeni u matematičkoj teoriji masovnog opsluživanja, još uvek ne postoji mogućnost da se reše svi praktični problemi. Često se dešava da sistemi bivaju uglavnom zasnovani na pretpostavkama pa ne prezentuju realni sistem u potpunosti, ili bivaju suviše složeni da bi se rešili matematičkim metodama[1].

Primena simulacionih metoda u rešavanju zadataka teorije masovnog opsluživanja zasniva se na istim teorijskim osnovama na kojima su zasnovane metode ocenjivanja u matematičkoj statistici. Nepoznata verovatnoća slučajnog događaja određuje se tako što se više puta ponavlja eksperiment u kome dati događaj može da se ponovi i za ocenu tražene verovatnoće se uzima relativna frekvencija tog događaja. Ogromna prednost korišćenja simulacionih metoda sastoji se i u tome što se eksperimenti izvode dovoljno brzo. Često je ipak potrebno dovoljno veliki broj ponavljanja eksperimenta da bi se tražena verovatnoća ili srednja vrednost mogle odrediti sa naznačenom tačnošću, pri čemu se ocene zasnivaju na zakonima velikih brojeva[1].

Simulacija podrazumeva veći broj različitih aktivnosti kojima se rešavaju pojedine grupe problema. Kada je koriste računarski stručnjaci, oni pod simulacijom podrazumevaju proces izgradnje apstraktnih modela koji predstavljaju neke sisteme ili podsisteme iz realnog sveta, kao i obavljanje određenog broja eksperimenata nad njima. Velika brzina rada računara omogućila je primenu metode simulacije za rešavanje brojnih složenih problema u najrazličitijim oblastima ljudskog rada i praktično doprinela naglom razvoju i njenoj sve većoj primeni[1].

Metodologija simulacije zasniva se na računarskim naukama, statistici, numeričkoj matematici i operacionim istraživanjima, ali je danas dovoljno razvijena da može biti posebna naučna disciplina. Simulacija nije obično izvršavanje računarskog programa, nije samo statistički test dobijenih rezultata niti optimizaciona procedura kao linearno programiranje.

Ona nam omogućava da definišemo i uočimo akcije i politike koje su dobre i ispravne. Simulacioni eksperimenti najčešće se izvode sa ciljem da se prikupe određene informacije, čije bi dobijanje putem eksperimenta nad samim sistemom bilo nepraktično ili suviše skupo. Te informacije se kasnije transformišu u odluke koje su značajne za upravljanje realnim sistemom koji je predmet simulacionog modeliranja.

Cilj simulacije jeste da prouči ponašanje sistema koji se simulira ali i da se ustanovi kako bi se isti sistem ponašao kada bi na njega delovao neki drugi skup promenljivih okolnosti. Može se postaviti pitanje zbog čega je potrebno jedan sistem zameniti modelom, a zatim vršiti simulaciju. Postoji više razloga, kao što su sledeći[3].

- Eksperiment nad realnim sistemom može da bude skup ili čak nemoguć;
- Analitički model nema analitičko rešenje;
- Sistem može da bude suviše složen da bi se opisao analitički;
- Eksperimentisanje sa realnim sistemom čak iako se zanemare drugi aspekti, uglavnom je neisplativo ili suviše složeno. Modeliranje, sa druge strane, može da ukaže na to da li je dalje ulaganje u eksperiment ekonomski opravdano ili ne;
- Izgradnja modela i simulacija ponekad imaju za cilj da se shvati funkcionisanje postojećeg sistema čija je struktura nepoznata i ne može joj se prići;
- Vreme može biti jak razlog da se pribegne simulaciji. Pri simulaciji vreme se može sažeti. To je značajno kod simulacije dugotrajnih procesa. U drugim slučajevima vreme se može značajno produžiti;
- Prilikom iznalaženja optimalnog funkcionisanja nekog sistema, uobičajeno je da se menjaju razni parametri. Često je to neizvodljivo sa realnim sistemom, bilo zato što takvog sistema uopšte nema ili zato što bi takav eksperiment bio preskup. Tada su gradnja modela i njegova simulacija moguće rešenje;
- Ponekad treba simulirati uslove pod kojima nastaje razaranje sistema. Naravno, razaranje realnog sistema najčešće nije dopustivo. U takvim prilikama simulacija predstavlja pravo rešenje.

Da bi se kreirao model potrebno je uraditi detaljnu, sistemsku analizu realnog sistema, prikupiti sve potrebne podatke i na osnovu njih vršiti modeliranje. Kada je simulacioni model završen, on se uz pomoć programskog jezika prevodi u simulacioni program. Na osnovu njega se vrše simulacioni eksperimenti i dobijaju podaci o ponašanju modela. Pošto je taj model apstraktni prikaz realnog sistema, na osnovu njegovog ponašanja se donose zaključci o ponašanju sistema.

2.2.1. Prednosti i nedostaci simulacije

Korišćenje simulacionih metoda zahteva poznavanje njihovih prednosti i nedostaka. Osnovne prednosti simulacije su sledeće[3]:

- Simulacione metode se koriste kao pomoć kod analize i donošenja odluka;
- Omogućeno je praćenje brzih promena u sistemu;
- Jednom izgrađen model se može višestruko koristiti;
- Omogućeno je veće i bolje razumevanje sistema;
- Simulacione metode je lakše primeniti nego analitičke metode;
- Podaci dobijeni ovim putem mogu se dobiti jeftinije od sličnih podataka dobijenih iz realnog sistema;

- Dovodi do smanjenja rizika;
- Smanjuje se vreme potrebno za izvršavanje eksperimenata.

Nedostaci simulacije su sledeći:

- Simulacioni modeli za digitalne računare mogu biti skupi i zahtevati mnogo vremena za njihovu konstrukciju i validaciju;
- Ponekad se koriste i kad analitičke tehnike daju zadovoljavajuće rezultate;
- Ne dobijaju se zavisnosti izlaznih promenljivih od ulaznih promenljivih modela niti optimalna rešenja;
- Zbog statističkog karaktera simulacije, potrebno je izvođenje većeg broja simulacionih eksperimenata kako bi se dobio odgovarajući uzorak rezultata simulacije, a već i pojedinačno izvođenje eksperimenata može zahtevati dosta vremena i memorije računara.

2.2.2. Simulacioni proces

Simulacioni proces je struktura rešavanja stvarnih problema pomoću simulacionog modeliranja. On se može prikazati u obliku niza koraka koji opisuju pojedine faze rešavanja problema ovom metodom[3].

Struktura simulacije nije strogo sekvencijalna već je moguć i povratak na prethodne korake procesa zavisno od rezultata dobijenih u pojedinim fazama procesa. Broj faza i redosled njihovog obavljanja zavisi od svake konkretne situacije, ali je ipak moguće navesti jedan opšti, uređen skup procedura. Osnovni koraci simulacionog procesa mogu se videti na slici 2.2 i biće navedeni u nastavku[3].

1. *Definisanje cilja istraživanja*

U ovom koraku se vrši definisanje željenog cilja i svrhe studije. Utvrđuje se problem koji treba rešiti (oblikovanje sistema, analiza sistema i sl.), granice sistem-okolina, nivo detaljnosti. Cilj istraživanja u ovom redu jeste određivanje nivoa kvaliteta opsluge korisnika poštanskih usluga u jedinici poštanske mreže. Svrha te analize je davanje preporuka za promenu postojeće organizacije kojima bi se postigao bolji kvalitet opsluge uz bolju iskorišćenost sistema

2. *Identifikacija sistema*

Ovaj korak podrazumeva opis komponenti sistema, utvrđivanje interakcije komponenti, načina rada, veze sa okolinom kao i formalni prikaz sistema.

3. *Prikupljanje podataka o sistemu i njihova analiza*

U ovoj fazi se vrši prikupljanje i merenje relevantnih podataka o sistemu, analiza tih podataka - izbor raspodela slučajnih promenljivih, ocena vrednosti parametara raspodele itd. Prikupljanje podataka o sistemu vršeno je u toku pet radnih dana (počevši 8.10.2014. do 14.10.2014.) u tačno određenim vremenskim intervalima i to u toku časa najvećeg opterećenja. Vršeno je merenje saobraćaja i vremena opsluge korisnika sortiranih u grupe prema sličnosti vremena potrebnog za obavljanje usluga, odnosno vrsti zahtevane usluge.

4. Izgradnja simulacionog modela

Ovaj korak podrazumeva stvaranje konceptualnog modela koji adekvatno opisuje sistem i omogućava rešavanje zadatog problema.

5. *Izgradnja simulacionog programa*

Tu se podrazumeva izbor programskog jezika ili paketa za stvaranje simulacionog programa pisanjem programa ili automatski generisanjem programa na osnovu konceptualnog modela.

6. *Verifikacija simulacionog programa*

Ovaj korak podrazumeva testiranje simulacionog programa prema pretpostavkama simulacionog modela. Proces verifikacije treba da pokaže da li je i u kojoj meri konceptualni model na odgovarajući način predstavljen računarskim kodom, odnosno u kojoj meri se slažu konceptualni model i računarski kod. Proces testiranja ispravnosti jednog modela najčešće prepliće procese verifikacije i validacije, pa zato ne postoji standardni način za proveru. Za to se koriste sledeće vrste provera:

- Ručna verifikacija logičke ispravnosti: model se izvesno vreme propušta na računaru i ručno, a potom se porede dobijeni rezultati;
- Modularno testiranje: pojedinačno testiranje svakog modula kako bi se ustanovilo da li program daje razumne izlaze za sve moguće ulaze;
- Provera u odnosu na poznata rešenja: podesimo model tako da predstavlja sistem čija su rešenja poznata i upoređujemo ih sa rezultatima modela;
- Testiranje osetljivosti: variramo jedan parametar dok ostali ostaju nepromenjeni i proveravamo da li je ponašanje modela osetljivo na promenu tog parametra;
- Testiranje na poremećaje: postavljamo parametre modela na neprirodne vrednosti i proveravamo da li se model ponaša ne neshvatljiv način. Na taj način je moguće otkriti greške u programu koje je veoma teško uočiti na drugi način.

7. *Validacija simulacionog modela*

Ispitivanje da li simulacioni model adekvatno predstavlja realni sistem. Ukoliko vrednovanje modela nije uspelo, potrebno je vratiti se na 4. korak.

8. *Planiranje simulacionih eksperimenata i njihovo izvođenje*

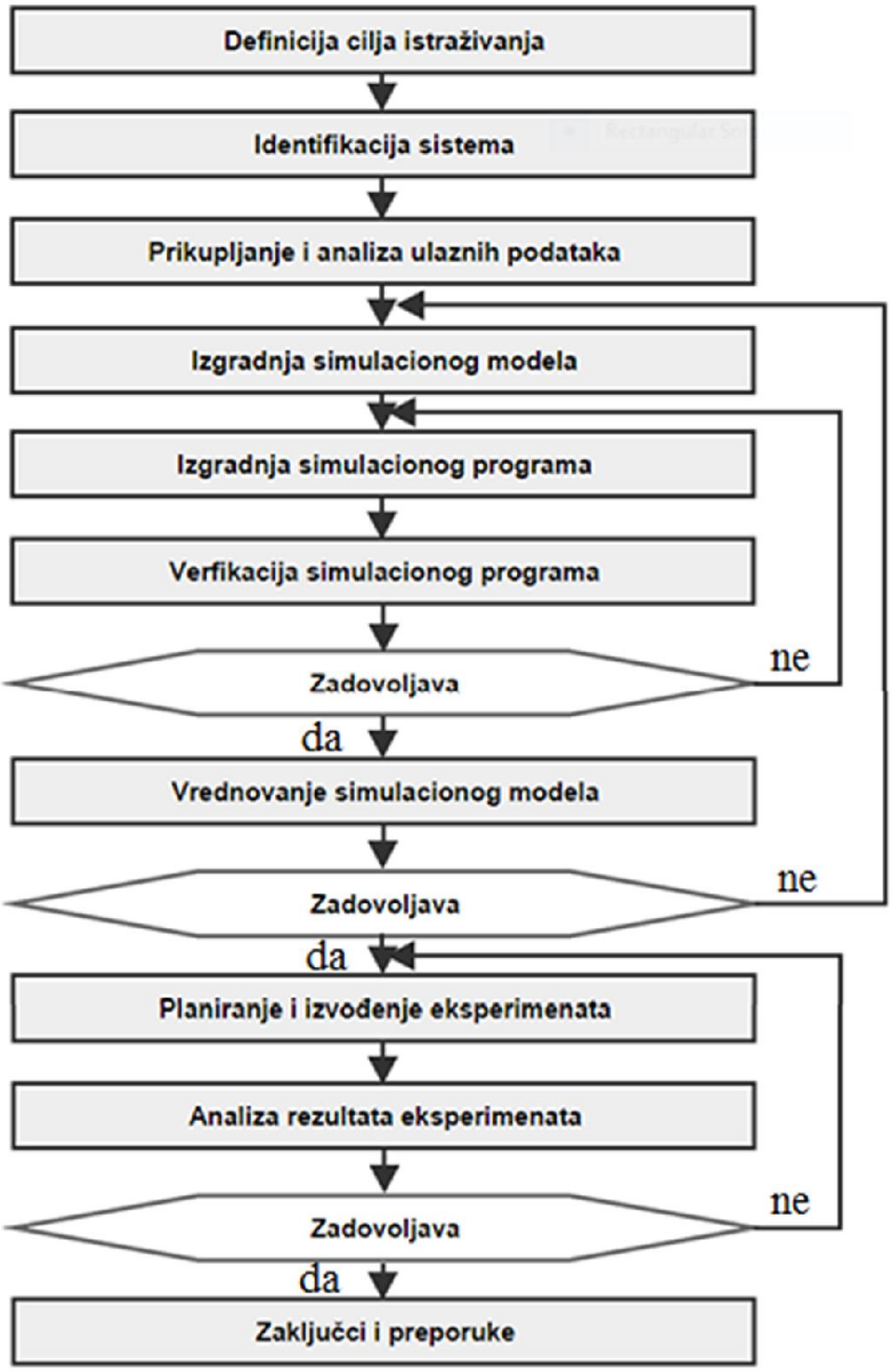
Ovo je veoma bitan korak u kome se planiraju simulacioni eksperimenti kojima se dolazi do ispunjenja cilja studije. Tu se podrazumeva planiranje promene parametara, ponavljanje eksperimenata zbog analize uticaja slučajnih promenljivih itd. Eksperimenti se izvode prema utvrđenom planu.

9. *Analiza razultata eksperimanata*

Statistička analiza rezultata simulacionih eksperimenata. Tokom analize rezultata može se ukazati potreba za dopunom koraka 8.

10. *Zaključci i preporuke*

Ovo je završni korak koji podrazumeva prezentovanje relevantnih razultata na osnovu kojih se mogu doneti odgovarajuće odluke, izbor konfiguracije sistema, izmene u sistemu i dr.



Slika 2.2 dijagram toka simulacionog procesa

2.2.3. Simulacija diskretnih događaja

Pod simulacijom diskretnih događaja podrazumeva se primena metode simulacionog modeliranja sistema kod kojih se javljaju diskretne promene stanja u sistemu ili njegovom okruženju. Te promene se menjaju diskontinualno u vremenu i mogu se predstaviti skupom događaja. Ova metoda simulacionog modeliranja se najčešće koristi za analizu dinamičkih sistema sa stohastičkim karakteristikama. Pod događajem se podrazumeva diskretna promena stanja entiteta sistema. Događaj nastupa u određenom trenutku vremena, što znači da se promene stanja entiteta menjaju u pojedinim trenucima vremena – kada nastupa događaj. Simulacija opisuje svaki diskretni događaj, krećući se od jednog događaja do drugog događaja pri čemu nastaje pomak vremena simulacije. Između dva uzastopna događaja, stanje sistema se ne menja[3].

Šalterski sistem pošte je tipičan primer stohastičkog sistema sa diskretnim događajima. U njemu se promenljive stanja, kao što je broj klijenata u šalter sali, broj opsluženih korisnika ili stanje na njihovim računima menjaju isključivo u određenim trenucima vremena i to kada klijent uđe u poštu, izvrši neku transakciju i izađe[3].

2.2.4. Vreme

Stanje modela sistema zasnovanog na diskretnoj stohastičkoj simulaciji menjaju događaji koji se odigravaju u diskretnim vremenskim trenucima. Svaki momenat u vremenu u kome se jedan ili više događaja odigravaju naziva se vremenskim trenutkom. Razlikujemo dva vremena simulatora, relativno i apsolutno. Relativno vreme simulacije je vreme od početka do završetka neke faze u toku simulacije (npr.radni rad) nakon čega se vreme resetuje. Apsolutno vreme simulacije je vreme od početka do kraja simulacije. Za strategiju koja je ovde korišćena vreme simulacije se uvek pomera iz vremena izvršenja prethodnog u vreme izvršenja prvog narednog događaja (mehanizam pomaka na naredni događaj)[5].

2.2.5. Entiteti i resursi

Objekti čije se aktivnosti modeliraju u simulacionom programu nazivaju se entiteti, koji su opisani atributima (svojstvima). Atributa može biti više, ali su dva osnovna, vreme odigravanja narednog događaja i naredni događaj. Resursi su objekti simulacionog modela čija je osnovna uloga da ograniče aktivnosti entiteta. Resursi imaju na raspolaganju jedno ili više mesta u koje primaju entitete. Entiteti zauzimaju i oslobađaju mesta u resursu[5].

Postoji razlika između permanentnih i privremenih (prelaznih) entiteta. Permanentni entiteti se kreiraju na početki simulacije i do kraja simulacije se ne uništavaju, dok privremeni entiteti se kreiraju po potrebi u toku simulacije i uništavaju se onda kada više nisu neophodni[5].

2.2.6. Događaj

Događaj je diskretna promena stanja entiteta u određenom vremenskom trenutku. Između dva događaja stanje sistema se ne menja. Razlikujemo dve vrste događaja[5]:

1. Bezuslovni događaj (planirani događaj) je onaj čije se odigravanje može predvideti i zato se unapred planira.

2. Uslovni događaj (neplanirani događaj) je onaj čije odigravanje zavisi od ispunjenja određenog uslova (npr.raspoloživost resursa).

Onda kada entitet učestvuje u nekom planiranom događaju, vreme odigravanja planiranog događaja se pamti u entitetu. Događaj se aktivira, onda kada je vreme nastupanja događaja jednako vremenu simulacije. Postoji jedna značajna vrsta planiranih događaja kojim se realizuju dolasci privremenih entiteta u sistem. Svaki put kada se taj događaj odigra, on stvara naredni entitet koji ulazi u sistem i postavlja vreme njegovog dolaska[5].

2.2.7. Stanje entiteta i redovi čekanja

Jednom kada se kreira entitet može da bude u jednom od sledećih stanja[5]:

1. Aktivan – događaj entiteta se upravo izvršava;
2. Planiran – događaj entiteta je planiran i entitet čeka da se izvrši;
3. Blokiran – događaj entiteta čeka da se izvrše sve dok se ne ispuni neki uslov;
4. Uništen – entitet je predviđen za uništavanje.

Entiteti koji se nalaze u blokiranom stanju smeštaju se u redove čekanja. Redovi čekanja mogu imati bilo koju disciplinu opsluživanja, ali najčešće su FIFO tipa[5].

2.2.8. Aktivnosti i grananje aktivnosti

Aktivnost čine skup događaja koji menjaju stanje entiteta. Obično započinju uslovnim događajima, a završavaju se planiranim graničnim događajem. Takve aktivnosti se nazivaju "zadržavanjem" i njihovo vreme trajanja nije unapred poznato. Nekada je vreme trajanja aktivnosti moguće i unapred planirati sa jednim ili više bezuslovnih događaja. U sistemu, entiteti ponekad treba da izaberu između različitih putanja. Grananje se može odigrati bilo iza planiranog ili uslovnog događaja. Postoje različiti kriterijumi za izbor rute[5]:

1. Izbor putanje je slučajan i ne može se unapred predvideti;
2. Izbor putanje zavisi od neke karakteristike entiteta;
3. Izbor putanje zavisi od stanja resursa, dužine reda čekanja, itd.

2.2.9. Događaji u simulacionoj strategiji raspoređivanja događaja

Kada su događaji u simulacionim strategijama u pitanju moguće je uspostaviti jedno pravilo, a to je da što je simulaciona strategija jednostavnija to su događaji složeniji. Tako, možemo reći da su za interakciju procesa koju je najteže realizovati događaji najjednostavniji, dok su događaji za raspoređivanje događaja najsloženiji. Strategija raspoređivanja događaja ima samo bezuslovne događaje tako da je u okviru tih događaja potrebno realizovati i ono što se odigrava u uslovnim događajima druge dve strategije. U bezuslovnom događaju koji kreiraju privremene entitete u modelu se, ukoliko entitet treba da zauzme neki resurs prvo taj entitet smešta u red čekanja, a nakon toga se proverava zauzeće resursa, oslobađanje iz reda i zauzimanje mesta ukoliko je slobodan i raspoređivanje za opslugu (algoritam 1). U tom događaju se kreira novi entitet i raspoređuje mu se vreme dolaska. Kod ostalih događaja je bitno naglasiti da ukoliko u prethodnom bezuslovnom događaju ima smeštanja u red čekanja i zauzimanja resursa, neophodno je da onaj entitet koji je izašao iz resursa kasnijem događaju oslobodi naredni entitet iz reda i da zauzme njegovo mesto u resursu (algoritam 2)[5].

```
{ Bezuslovni događaj }
procedure DolazakEntiteta
begin
	…
	Postavi resurs u red dačeka;
	if resurs raspoloživ then
		Primi prvi entitet iz reda;
		Zauzmi jedno mesto u resursu;
		Postavi vreme za kraj opsluge entiteta;
	end
	{ Kreiranje i raspoređivanje novog entiteta }
	Kreiraj sledeći entitet;
	Postavi vreme dolaska sledećeg entiteta;
end
```

Algoritam 1. događaj koji kreira privremene entitete u model

```
{ Bezuslovni događaj }
procedure NekiBezuslovniDogađaj
begin
	...
	{ Entitet oslobađa naredni entitet }
	if red nije prazan then
		Primi prvi entitet iz reda;
		Zauzmi jedno mesto u resursu;
		Postavi vreme za kraj opsluge entiteta;
	end
end
```

Algoritam 2. događaj sa tekućim entitetom koji oslobađa naredni entitet

2.3. Programski jezici za izvođenje simulacije

Postavlja se pitanje kako izabrati programski jezik kojim ćemo reprezentovati model. Simulaciju je moguće izvesti pomoću programskih jezika opšte namene, kao što su FORTRAN, PASCAL, BASIC, C jezici i drugi. Sa druge strane, model možemo opisati simulacionim programskim jezicima koji su specijalizovani za ovaj tip problema. Izbor jezika zavisi i od same simulacije. Većina programski jezika opšte namene, kao C++, su dobro standardizovani, imaju kompajlere za sve tipove računara i razvojno okruženje je dostupno na gotovo svim operativnim sistemima, dostupne su biblioteke sa gotovim programima i potprogramima, algoritmima i skladištima podataka. Međutim, simulacije napisane u programskim jezicima opšte namene, su relativno duge i glomazne. Svaka promena u simulacionom modelu često zahteva dosta dug vremenski period prepravljanja samog koda, a često isti može biti i napisan na najrazličitiji broj načina što otežava posao, onom koji se bavi modifikovanjem. Sa druge strane, simulacioni jezici su uglavnom namenjeni za rešavanje jedne klase problema i često se javlja potreba za nekim dodatnim uslovom, ili nekom dodatnom statistikom, takvom da je njeno realizovanje jako komplikovano ili čak nemoguće[4].

Simulacioni program je moguće realizovati u specijalizovanim simulacionim jezicima (GOSS, SIMSCRIPT II/III, SIMUL8...), simulacionim paketima(Arena, Automod, Flexim, Witness...) ili u nekom programskom jeziku opšte namene (Pascal,C,C++,Java, Python.).

U programskom jeziku moguće je iskoristiti neku od postojećih simulacionih biblioteka (sim(C++),CSIM19(C), baseSim(Delphi),C++SIM, JavaSim, SimPy(Python)) ili napraviti simulacioni program od početka[5].

GPSS – General Purpose Simulation System je jezik koji se naročito koristi za simulaciju modela sistema masovnog opsluživanja, a posebno kod sistema gde je matematička i statistička analiza veoma složena ili ne daje zadovoljavajuće rezultate. GPSS jezik je razvio IBM 1961. godine, a to je bila verzija GPSS – I.

Neke verzije ovog jezika su : GPSS/360, GPSS V,GPSS PC (verzija za personalne računare), GPSS/H(verzija za personalne računare pod UNIX ili DOS operativnim sistemom, ova verzija omogućava vezu iz FORTRAN i C jezika)[4].

U Beogradu postoji GPSS/FON verzija, razvijena na Fakultetu Organizacionih Nauka od strane profesora dr. Božidara Radenkovića. Ova verzija jezika GPSS realizovana je u PASCAL – u. Na njoj se rade vežbe na Saobraćajnom fakultetu[6].

3. SIMULACIJA U JEZICIMA GPSS I C++

3.1.Simulacioni jezik GPSS

GPSS predstavlja interpreterski jezik za simulaciju diskretnih – stohastičkih sistema. Struktura modela se definiše a simulacija izvršava pomoću naredbi ugrađenog jezika[3].

Nakon simulacije na raspolaganju su statistički pokazatelji o ponašanju modela u toku simulacije. GPSS je jezik orijentisan na procese, model se predstavlja dijagramom toka koji se konstruiše uz pomoć blokova. Svakom bloku odgovara jedna linija. Blokovi su statički objekti[3].

Entitete (objekte modela) možemo podeliti u četiri klase[3]:

1. **Dinamički enetiteti** – transakcije su dinamički objekti GPSS jezika koje se kreću kroz blokove. One imaju niz karakteristika koje nazivamo parametrima. One su aktivni objekti i može postojati više transakcija u svakom trenutku na različitim mestima u blok dijagramu. Predstavljaju saobraćajne jedinice u modelu(klijenti na opsluživanju, telefonski pozivi, vozila na servisu). Transakcije mogu da rezervišu određene resurse u modelu.
2. **Statički entiteti** – to su elementi opreme, pasivni su i predstavljaju resurse na koje se deluje transakcijama. Tu spadaju: uređaji (mogu da usluže samo jednu transakciju u nekom vremenskom trenutku), skladišta (mogu da usluže više transakcija istovremeno u zavisnosti od kapaciteta) i logički prekidači (stanje prekidača se postavlja prolaskom transakcije kroz njega, mogu biti u tru stanja, uključeno, isključeno i invertovano, služe za testiranje stanja opreme i zavisno od toga transakcija se preusmerava).
3. **Statistički entiteti** – u ovu grupu spadaju redovi i tabele. Ako je uređaj zauzet ili skladište puno tada transakcija formira red. Na osnovu statistike GPSS automatski formira tabele sa vrednostim. Korisnik može i sam da formira tabele sa statističkim vrednostima koje su mu potrebne. Ona sadrži frekvencijske klase sa numeričkim vrednostima nekog atributa.

4. **Entiteti operacija** – oni se nazivaju blokovima i pružaju logiku sistema, dajući instrukcije transakcijama gde treba da idu i šta dalje da rade. Blok dijagram predstavlja operacije koje se vrše unutar datog sistema. Blok pokazuje tip operacije koju obavlja a linije između blokova ukazuju na tok saobraćaja. Unutar blok dijagrama mogu da nastanu četiri osnovna tipa događaja:
   a) Stvaranje ili uništavanje transakcija;
   b) Promena numeričkih atributa entiteta;
   c) Kašnjenje transakcije za određeno vreme;
   d) Promena toka transakcije kroz blok dijagram.

3.1.1. Osnovni koncept GPSS jezika

Transakcije generiše blok GENERATE. Ona se kreće kroz model sve dok ne naiđe na blok koji nema uslov da je primi ili na naiđe na blok TERMINATE koji uklanja transakciju iz modela. Takođe, postoje blokovi koji zadržavaju transakciju za određeni period simulacionog vremena. Ako neki blok nema uslova da primi transakciju, onda ona čeka da se ispuni uslov daljeg kretanja kroz model. Prolaz transakcija kroz određene blokove izaziva promene koje utiču na stanje modela i njegovo okruženje[3].

Transakcije u GPSS – u se čuvaju u listi tekićih događaja LTD i listi budućih događaja LBD. Transakcije iz LTD nastoje da se kreću kroz blok dijagram. Transakcije iz LBD nisu spremne za kretanje, već kasnije, kada se ažurira simulacioni sat (ispunjeni uslov kretanja), ove transakcije se prebacuju u LTD kako bi nastavile kretanje kroz model. U svakoj tački simulacionih vremena (simulacioni sat zadat numeričkim atributom C1 uzima samo celobrojne vrednosti), GPSS skanira sve transakcije u LTD i svaka koja ima uslov daljeg kretanja to i čini. Kretanje transakcija se prekida zbog jednog od tri razloga[3]:

1. Transakcija je uništena;
2. Blok odbija da primi transakciju;
3. Transakcija je ušla u blok koji je zadržava izvestan period simulacionog vremena.

Ukoliko je u pitanju treći razlog, transakcija se prebacuje is LTD u LBD. Ukoliko je kretanje transakcije izazvalo promene stanja modela, GPSS ponovo obavlja skaniranje, ali od početka LTD, u suprotnom naredna transakcija iz LTD se skanira[3].

Kada nastupi trenutak da nijedna transakcija iz LTD ne može da se kreće, GPSS ispituje LBD. LBD je uređena tako da se na njenom početku nalaze transakcije koje uslov za kretanje kroz model stiču ranije, dok su na kraju one transakcije koje moraju duže da čekaju. Kada nijedna transakcija iz LTD ne može da nastavi kretanje, GPSS ažurira simulacioni sat na vreme prve transakcije iz LBD. Sve transakcije iz te liste, koje su sada stekle uslov za kretanje, kopiraju se iz LBD u LTD i skaniranje LTD se ponavlja.

Svaka transakcija ima svoj prioritet, koji se može menjati sa protokom simulacionog vremena i trenutnim položajem transakcije u blok dijagramu. LTD je sortirana po opadajućem redosledu prioriteta, tako da se transakcije sa većim prioritetom kreću pre onih čiji je prioritet manji u svakom trenutku simulacionog vremena[3].

3.1.2. Vrste naredbi u GPSS – u

1. Deklaracione naredne entiteta definišu atributa pojedinih permanentnih entiteta u programu. Tu spadaju naredbe za deklaraciju uređaja, skladišta, tabela (histograma), funkcija i dr. Mogu se pisati bilo gde ali je uobičajeno da se pišu na početku programa. U polju LOKACIJE specifira se posebno definisano skladište, tabela i slično a u polju VARIJABLE mogu da budu upisani različiti argumenti.
2. Blok naredbe služe za specifikaciju modela sistema. One se izvršavaju kada transakcija prilikom kretanja kroz model naiđe na blok. Model se sastoji od niza blok naredbi povezanih u obliku blok dijagrama. Svaka blok naredba može da ima identifikacioni broj, koji se piše u polju lokacije. Ako korisnik želi da se pozove na neki blok on može da u polje LOKACIJE unese određeni mnemonički simbol. U polju OPERACIJE piše se tip blok naredbe, a u polju VARIJABLE pišu se argumenti.
3. Kontrolne naredbe služe za kontrolu izvršenja simulacije, a mogu imati uticaj na statistiku o ponašanju entiteta u toku simulacije. Simulator zahteva neke informacije upravljanja kao što su dužina simulacije, kada je kraj naredbi programa i drugo. U polju OPERACIJA piše se tip naredbe, u polju VARIJABLE unose se argumenti specifirane naredbe a polje LOKACIJE se ne koristi. Tu spadaju naredbe SIMULATE,START, RESET, CLEAR, END[3].

Trajanje simulacije može biti ograničeno brojem transakcija koje su prošle kroz model, što se kontroliše pogodnom upotrebom naredbi START i TERMINATE ili vremenski (fiksno), što je određeno takozvanim tajmerom koji čini par naredbi GENERATE i TERMINATE [3].

3.2.Objektno orijentisana simulacija

Objektno orijentisana simulacija (OOS) je simulacija koja koristi koncept objekata koji se sastoje od podataka sa jedne strane i opisa ponašanja sa druge strane. Kreiraju se šabloni koji predstavljaju uopštenu sliku nekog dela podsistema ili celog sistema određenog simulacionog modela. Na primer, takav jedan sistem može predstavljati univerzalni poštanski šalter koji vrši ulogu opsluživanja korisnika u pošti. Šablon se kreira tako da se u napred predvide veličine i osobine koje bi ovakav sistem zahtevao. Sa druge strane i sam korisnik može biti predstavljen nekim novim šablonom, sa nekim novim karakteristikama. Ovakav pristup simulaciji omogućava da se sa istog šablona kreiraju različiti objekti koji će imati iste ili slične osobine kao i sam šablon, ponašaće se slično i u modelu će biti realizovan na sličan način. Drugim rečima to znači, da iz istog šablona za univerzalni šalter možemo napraviti dva potpuno različita objekta sa drugačijim vremenima opsluge. Međutim, iako različita, ova dva objekta će se slično ponašati u smislu opsluge korisnika. Na ovaj način, jednostavnim kreiranjem i uništavanjem objekata mi možemo kreirati novog korisnika, novi šalter, sa potpuno jedinstvenim karakteristikama a sličnim ponašanjem. Terminologija je ovde namerno izostavljena radi lakšeg razumevanja samog pristupa OOS.

3.2.1. Programski jezik C++

C++ je jezik *opšte namene, srednjeg nivoa apstrakcije,* podržava *objektno orijentisano, proceduralno i generičko programiranje.* To je jezik koji se kompajlira, jezik koji pravi razlike između malih i velikih slova i jezik slobodne forme.

C++ je takođe jezik statičkog tipa. Ovaka opis programskog jezika programerima daje okvirnu sliku o njegovoj složenosti, pravilima pisanja i mogućnostima[8-10].

Kada kažemo da je programski jezik C++, jezik opšte namene, mislimo na to da se taj jezik može koristiti za kreiranje aplikacija u najrazličitijem broju domena, drugim rečima, to znači da C++ jezik može biti korišćen za izradu aplikacija koje bi se bavile upravljanjem hardverskih komponenti, animacijom, simulacijom, izradom teksta, izradom internet stranica, audio i video tehnologijama, video igricama mnogim drugim aplikacijama[8-10].

C++ je jezik srednjeg nivoa apstrakcije iako pruža mogućnost i nižih i viših nivoa. Da bi objasnili šta znači srednji nivo apstrakcije prvo ćemo objasniti niži i viši nivo apstrakcije. Niži nivo apstrakcije (eng.low-level language) je programski jezik koji je blizak mašinskom jeziku, udaljen je od korisnika u smislu razumljivosti i načina na koji se jezik piše ili čita.

Niži programski jezici omogućavaju direktan pristup memoriji, brzi su u pogledu izvršavanja instrukcija i koriste se za pisanje softvera koji upravlja hardverom, kao na primer operativni sistem ili se koriste za uzgradnju kompajlera jezika koji se nalazi na višim nivoima.

Sa druge strane, jezici višeg nivoa apstrakcije su jezici koji su jako bliski korisnicima. Kod koji se piše je razumljiv, same komande su najčešće na engleskom jeziku i razumljive su. To su jezici koji često imaju ugrađene algoritme koji vode računa o zauzetoj memoriji kako ne bi došlo do njenog potpunog zauzimanja nekim podacima koji se više ne koriste i time usporavali sam računar koji izvršava program.

Konačno, jezici srednjeg nivoa apstrakcije predstavljaju nešto između, u smislu da je kod dovoljno jasan i razumljiv za kreiranje i održavanje složenih aplikacija, ali isto tako sam apstrakcija nije gurnuta do krajnjih granica. Drugim rečima, ako bi uporedili programske jezike C++ i Ruby(jezik višeg nivoa) mogli bi reći da su oba jezika objektno orijentisana, međutim drugi je u ovom slučaju potpuno objektno orijentisan jezik i sve se može posmatrati kao objekat, dok sa struge strane u C++ jeziku postoje osnovni ugrađeni tipovi podataka koji se predstavljaju direktno u memoriji, kao što su celi brojevi(eng.integer, int) , realni (double) itd[6].

U klasičnom programiranjju, kod i podaci su odvojeni. Objektno orijentisano programiranje menja taj odnos i za razliku od struktuiranog programiranja, pruža mogućnost korisnicima da pakuju kod i podatke u formu klasa(eng.*clas*s) . U tom smislu klase mogu imati neko stanje ili vrednost (*data fields - attributes*) i ponašanje (*proceduress - methods*). Kad klasu jednom deklarišemo, možemo je koristiti za formiranje pojedinačnih stavki podataka – objekata (*instance*). Svaki objekat ili instanca klase ima sopstvene vrednosti ali ima zajedničko ponašanje koje je programirano u klasi[8-10].

Generičko programiranje se odnosi na vrstu programiranja gde se algoritmi i kontejneri podataka pišu tako da se tip podataka koji koriste specifira kasnije. Ovo pruža programerima više mogućnosti prilikom pisanja programa, jer isti algoritam mogu primeniti na više različitih kontejnera[8-10].

Za razliku od GPSS koji interpreterski jezik, C++ je kompajlerski jezik, a to u najjednostavnijem smislu znači da program napisan u C++ jeziku mora prvo biti kompajliran od strane kompajlera, zatim se vrši linkovanje, sastavljanje i na kraju se izvršava, dok GPSS program izvršava naredbu po naredbu[6].

Programski jezik slobodnog tipa je onaj jezik u kome pozicija ispisanog teksta, programski koda, na ekranu ne utiče na program. Drugim rečima, lokacija karaktera na strani programskog teksta nije značajna[8-10].

Statički i dinamički tip programskog jezika se odnosi na to da, u slučaju statičkog programiranja tipovi podataka su zadani u fazi kompajliranja, dok se kod dinamičkog tipa programiranja, tipovi podataka zadaju u fazi izvršenja programa (*run - time*). Ovako dat opis svrstava C++ u obe kategorije, ali se po standardizaciji C++ svrstava u statički tip[8-10].

C++ jezik je kreirao profesor **Bjorne Stroustrup** 1979.godine. Baziran je na jeziku C i prvobitno nazvan C sa klasam, a 1983.godine dobija naziv C++[6].

3.2.2. C++ kao objektno orijentisan jezik

Objektno orijentisano programiranje (OOP) je deo objektne paradigme koja obuhvata osnovne objektne koncepte[6]:

1. Apstraktni tipovi podataka (*eng. abstract data types*), tip koji je definisao programer, za koji se mogu kreirati instance i koji je predstavljen strukturom i ponašanjem;
2. Enkapsulacija (*eng. encapsulation*), deo softvera ima jasan definisan interfejs i implementaciju, interfejs je svima dostupan, implementacija je nedostupna;
3. Nasleđivanje (*eng. inheritance*), jedan tip može da nasledi drugi, sa značenjem da su njegovi objekti jedna vrsta osnovnog tipa;
4. Polimorfizam (*eng. polymorphizm*), isti interfejs se koristi za različite tipove podataka;

Kada govorimo o OOP, neizostavni elementi su klasa i struktura. Oba elementa su jako slična. Klasa je osnovna organizaciona jedinica programa u OOP jezicima.

Klasa predstavlja konstrukciju u koju su grupisani podaci (promenljive) i funkcije. Klasom se definiše novi korisnički tip za koji se mogu kreirati instance. Klasa može da predstavlja realizaciju apstrakcije iz domena problema. Svaki objekat (instanca) ima sopstvene elemente koji su navedeni u deklaraciji klase.

Ovi elementi klase nazivaju se članovi klase (*eng. Class members*). Članovi klase dostupni spolja (iz druge klase ili funkcije) nazivaju se javnim članovima (*eng. public*). Sa druge strane, članovi koji su nedostupni korisnicima klase (ali ne i članovima klase) nazivaju se privatnim (*eng. private*)[6].

Da bi objekat bio inicijalizovan, u klasi se definiše posebna funkcija koja se implicitno (automatski) poziva kada se objekat poziva. Ova funkcija se naziva konstruktor (*eng. constructor*) i nosi ime kao i klasa. Moguće je definisati i funkciju koja se poziva uvek kada objekat prestaje da živi. Ova funkcija naziva se destruktor(*eng. destrctcor*)[6].

Nasleđivanje je osobina klase da nasleđuje članove neke druge klase. Nasleđivanjem, članice zadržavaju dostupnost koja je postojala i u roditeljskoj klasi prema korisnicima klase.

Funkcija članica koja ća u izvedenim klasama imati nove verzije, deklariše se u osnovnoj klasi kao virtuelna funkcija (*eng. virtual function*).

Izvedena klasa može da da svoju definiciju virtuelne funkcije, ali i ne mora. Apstraktna klasa je klasa koja sadrži makar jednu apstraktnu, čistu virtuelnu funkciju,tj. virtuelnu funkciju čija je deklaracija izjednačena sa 0. Ona klasa kod koje su sve funkcije apstrakne naziva se interfejsom. Naziva se tako jer je to klasa koja izlaže svoj interfejs a ne i impementaciju. Zbog toga ne mogu da se prave objekti od apstraktnih klasa, već samo od klasa koje ih nasleđuju i implementiraju ponuđeni interfejs[6].

Svojstvo da se odaziva prava verzija funkcije klase čiji su naslednici dali nove verzije naziva se polimorfizam. Ostvaruje se korišćenjem virtualnih funkcija koje se preklapaju(*eng. overriding*)[6].

Šablonske klase(*eng. template*) su klase koje koriste jedan ili više nespecifiranih tipova. Umesto da se pravi više klasa, pravi se jedna klasa koja onda može da se koristi sa više različitih tipova. Na ovaj način je, takođe moguće ostvariti svojstvo polimorfizma[6].

3.3.Sličnosti i razlike C++ i GPSS jezika

Za razliku od GPSS jezika koji se koristi blokovima kako bi se formirao model, C++ je kao što je već rečeno, jezik opšte namene i nema tačno definisan kriterijum po kome bi se model izradio.

U objektno orijentisanom programiranju možemo da kreiramo novi, korisničko-definisani tip, koji možemo da poistovetimo sa ugrađenim tipovima poput realnih, celobrojnih podataka. To ostvarujemo preko klasa koje korisnik sam definiše u zavisnosti od toga kakve karakteristike zahteva.

Na konkretnom primeru objektno orijentisanog programiranja možemo objasniti korišćenje klase.

```
#include<iostream>
// Naredbom using ukljucujemo ceo prostor imena.
using namespace std;
class tacka{
// Kljucna rec public oznacava da je kod ispod javan
public:
// Atributi klase tacka x i y celobrojnog su tipa.
int x,y;
};
int main(){
// Kreiramo instancu (objekat) klase tacka i nazivamo je t.
tacka t;
// Dodeljujemo vrednost atributu x
t.x = 10;
// Dodeljujemo vrednost atributu y
t.y = 20;
// Ispisujemo vrednosti atributa x i y, objekta t, klase tacka.
cout << "t.x = " << t.x << "\nt.y = " << t.y;
return 0;
}
```

Da bi kod bio što kraći i razumljiviji za sada su korišćeni samo podaci klase, a rečeno je da klase mogu imati u svom opisu i kod koji se odnosi na zajedničko ponašanje objekata koje kreiramo iz iste klase. Na primeru to možemo objasniti kroz dve veoma važne metode klase koje su podrazumevane i svaka klasa ih ima.

Kompajler automatski generiše ove dve metode i u slučaju da ih korisnik ignoriše. Reč je o konstruktoru i destruktoru klase. Funkcije koje se pozivaju prilikom kreiranja i uništavanje instance. C++ nam nudi mogućnost pisanja naših jedinstvenih metoda koje će u tom slučaju biti korišćene umesto podrazumevanih[6].

```
#include<iostream>
// Naredbom using ukljucujemo ceo prostor imena.
using namespace std;
class tacka{
// Kljucna rec public oznacava da je kod ispod javan i direktno dostupan
public:
// Konstruktor je javan, i vrsi inicijalizaciju podataka instance.
tacka(int xx, int yy){
x = xx;
y = yy;
cout << "Objekat kreiran!" << endl;
}
// Destruktor klase
~tacka(){ cout << "Objekat unisten!" << endl; }
// Atributi klase tacka x i y celobrojnog su tipa.
int x,y;
};
int main(){
// Kreiramo instancu (objekat) klase tacka pozivajuci kreirani
konstruktor i nazivamo je t.
tacka t(10,20);
// Ispisujemo vrednosti atributa x i y, objekta t, klase tacka.
cout << "t.x = " << t.x << "\nt.y = " << t.y << endl;
return 0;
}
```

Navedeni primer je potpuno isti kao i prethodni sa tom razlikom što sada inicijalizaciju instance klase tačka vršimo pri samom kreiranju objekata tako što vrednosti `t.x` i `t.y` inicijalizujemo prosleđivanjem parametara 10 i 20. Takođe, dodate su poruke koje se ispisuju na ekran u trenutku kreiranja i uništenja objekata. Napomenimo da nareda *endl* predstavlja novi red i možemo reći da ima istu ulogu kao i \n (međutim i tu postoji razlika). Postavlja se pitanje šta se to ispisuje na konzoli po pokretanju programa. Jasno je da se objekat kreira, ispisuju se vrednosti podataka, međutim ni u jednom trenutku nija napisana naredba uništenja objekata u samom kodu[6].

Da li je ovde reč o takozvanom problemu curenja memorije (*eng. Memory leak*) koji je spomenut prilikom objašnjenja nivoa programskih jezika? Da bi objasnili koncept memorije koju program koristi objasnićemo šta su to stek (*stack*) i hip(*heap*). Tokom kreiranja aplikacije, zauzima se određeni deo memorije, jedan deo odlazi na sam kod koji se izvršava, jedan deo odlazi na globalne i statičke promenljive i jedan deo odlazi na stek. Stek je onaj deo memorije koje funkcije koriste za svoje izvršavanje. Tu spadaju sve promenljive koje funkcija koristi. U konkretno objašnjenom slučaju, objekat klase tačka biće kreiran unutar main funkcije i kao takav biće kreiran na steku. Sva memorija alocirana tj. zauzeta na steku po izvršenju date funkcije biva oslobođena tj. vraćena operativnom sistemu.

Drugim rečima, po završetku funkcije main(), pošto je instanca tačke kreirana na steku poziva se destruktor klase kako bi se izvršilo uništenje instance i oslobođenje memorije. U slučaju da je konstruktor u ovom slučaju definisan kao privatni, on ne bi bio dostupan za pozivanje i kompajler bi javio grešku. Ovo ujedno i daje odgovor na pitanje da li se javlja curenje memorije,a odgovor je ne. Stvari se komplikuju ako u razmatranje uključimo sada i dinamičku memoriju tj. hip[6].

Ponekad su klase veoma velike, pa samo kreiranje objekata zahteva veliki memorijski prostor koji nije predviđen stekom ili je potrebno sačuvati podatke o nekom objektu ili nekoj promenljivoj i po završetku neke funkcije.
U tim slučajevim koristimo dinamičku memoriju. C++ nam u tu svrhu predstavlja pokazivače (*pointers*) i njihova primena je gotovo ključna u skoro svim aplikacijama kreirane C++ jezikom. Pokazivači su ogromna prednost C++ jezika i ceo naš model se temelji na njihovoj primeni. Dalje se pokazuje primena pokazivača[6].

```
int main(){
// Kreiramo instancu (objekat) klase tacka i nazivamo je t.
tacka* t = new tacka(10,20);
// Ispisujemo vrednosti atributa x i y, objekta t, klase tacka.
cout << "t.x = " << t->x << "\nt.y = " << t->y << endl;
return 0;
}
```

U ovom slučaju je nebitno da li je konstruktor javan ili privatan, jer isti neće biti pozvan od strane samog programa po završetku izvršenja funkcije main i u ovom slučaju će se javiti problem curenja memorije. Napomenimo, da ne bude zabune, da stek ne funkcioniše tačno po principu jedan stek jedna funkcija, kako je objašnjeno, već je ceo pristup malo drugačiji. Kako se ne bi upuštali u objašnjavanje funkcionisanja raspodela memorija kazaćemo da je određeni deo prostora (*eng.scope*) zadužen za određeni deo memorije[8-10].

```
int main(){
// Kreiranje objekta t
tacka t(10,20);
{
// Kreiranje objekta t2
tacka t2(5,-1);
// Unistenje objekta t2
}
cout << "t.x = " << t.x << "\nt.y = " << t.y << endl;
cout << "t2.x = " << t2.x << "\nt2.y = " << t2.y << endl; //
Greska, t2 ne postoji u ovom delu programa
return 0; // Unistenje objekta t
}
```

3.4.Vek objekata u programskom jeziku C++

Svaki objekat u programskom jeziku C++ prolazi, redom, kroz sledeće faze života[7]:

1. Alokacija memorije;
2. Inicijalizacija;
3. Upotreba;
4. Deinicijalizacija;
5. Oslobađanje memorije.

Prema načinu pravljenja i uništavanja objekata, svi objekti u programskom jeziku C ++ se dele na statičke, automatske i dinamičke[7].

- Statički objekti se prave u postupku učitavanja programa u memoriji, pre pozivanja funkcije main. Redosled njihovog pravljenja zavisi kako od redosleda njihovog navođenja u tekstu programa, tako i od redosleda i načina povezivanja modula u izvršnu verziju programa. Često je veoma komplikovano izvoditi zaključke o redosledu pravljenja statičkih objekata, zbog čega se preporučuje izbegavanje koda koji zavisi od tog redosleda. Statički objekti se uništavaju po izlasku iz funkcije main, u redosledu obrnutom od redosleda pravljenja. U statičke objekte spadaju svi globalno definisani objekti i statički podaci.
- Svi lokalno definisani objekti predstavljaju tzv. automatske objekte. Pored njih, automatske objekte predstavljau i neimenovani preivremeni objekti. Automatski objekti se prave kada se pri izvršavanju koda dođe do definicije promenljive čiji sadržaj predstavlja novi objekat. Zapravo, nije pogrešno definiciju lokalne promenljive smatrati za izvršnu naredbu, jer ona u kodu vrlo precizno određuje i mesto i trenutak pravljenja automatskog objekta. Automatski objekti se uništavaju neposredno pre izlaska iz bloka u kome su definisani.
- Dinamički objekti su oni koji se prave primenom operator *new*. Za razliku od statičkih i automatskih objekata čije je trajanje precizno definisano lokalnim segmentom koda, trajanje dinamičkih objekata je teže opisati.
- Kao što se eksplicitno prave primenom operatora *new*, dinamički objekti se i uklanjaju eksplicitno – primenom operatora *delete*[7].

Pogledajmo sledeći primer i prodiskutujmo redosled pravljenja i uništavanja objekata[7]:

```
int a(1);
int main(){
//...
      int b(2);
      int *c= new int(3);
//...
      delete c;
}
```

Redosled pravljenja i uništavanja bi bio sledeći:

1. Pri učitavanju programa, automatski se pravi statički objekata a,
2. Poziva se funkcija main,
3. Izvršava se kod kojim je definisana funkcija main, sve do definicije promenljive b,
4. Pravi se automatski objekat b,
5. Eksplicitno se pravi dinamički objekat c,
6. Izvršava se kod funkcije main,
7. Eksplicitno se uništava dinamički objekat c,
8. Neposredno pre uništavanja bloka, automatski se uništava automatski objekat b,
9. Po završetku funkcije main, pri izbacivanju programa iz memorije, automatski se uništava objekat a.

U sva tri slučaja konstrukcija (inicijalizacija) se obavlja na potpuno isti način – konstruktorom sa jednim celobrojnim argumentom. Takođe, u sva tri slučaja deinicijalizacija se obavlja na potpuno isti način – podrazumevani destruktor[7].

Klase se ne definišu na osnovu sadržaja, već na osnovu ponašanja. Na osnovu zahtevanog ponašanja klase definiše se *interfejs klase.* Interfejs klase predstavlja sredstvo za upotrebu objekata klase. Čine ga deklaracije javnih metoda i podataka klase. Zbog težnje da se funkcionalnost u načinu upotrebe klase što više razdvoje od implementacije, a podaci predstavljaju upravo strukturni deo implementacije, preporučuje se da se podaci ne uključuju u interfejs klasa, tj.da ne budu javni.

Oblikovanje interfejsa počiva na analizi zahteva[7]:

- Za svaku zahtevanu operaciju se dodaje po odgovarajući metod (eventualno jedan metod može odgovarati većem broju srodnih operacija);
- Nazivi metoda se određuju odgovaranjem na pitanje „Šta metod radi?“
- Argumenti ili rezultat metoda se određuje na osnovu semantike metoda (eventualno se deo parametara može zameniti upotrebom podataka objekta ili izračunavanjem metoda koji opisuje stanje objekata koji primenjuje metod);
- Da bi klasa mogla ispravno da funkcioniše može biti neophodno da se interfejs proširi još nekim operacijama koje nisu eksplicitno zahtevane.

Nakon definicije interfejsa klase, prelazi se na njenu implementaciju. Pod implementacijom klase podrazumevamo definisanje tela metoda koji čine interfejs, kao i definisanje neophodnih privatnih metoda i članova. Implementacija može dovesti do manjih ili većih izmena u interfejsu, kako bi on postao kompaktniji ili bolje zaokružen. Ako se na samom početku posveti dovoljno pažnje oblikovanju interfejsa, takve izmene su relativno retke. [7]

3.5. Apstraktni tipovi kontejnera

Razlikujemo sekvencijalne i asocijativne kontejnere. Sekvencijalni kontejner sadrži uređenu kolekciju elemenata nekog tipa. U osnovne sekvencijalne kontejnere možemo da ubrojimo „vector“, „list“, „deque“. Asocijativni kontejneri su karakteristični po tome što efikasno vrše proveru prisustva kao i izdvajanje pojedinih elemenata. Njihovi elementi su sortirani po ključu, odnosno, svaki element je povezan (asociran) sa ključem.

U tom smislu, asocijativni kontejneri su bliski običnim nizovima: kod nizova svaki element ima indeks preko kojeg možemo da mu pristupimo. Kod asocijativnih kontejnera indeksi se nazivaju ključevima, a razlika u odnosu na nizove je u tome što ključevi ne moraju da budu celobrojnog, već mogu da budu bilo kog tipa[12].

3.5.1. Iteratori

Pod iteratorom podrazumevamo opšti metod pomoću koga pristupamo nekom elementu unutar bilo kog tipa kontejnera. To je pokazivač na element u kontejneru. Npr. ako je ***iter*** iterator, onda ***iter***++ pomera iterator unapred tako da adresira sledeći element u kontejneru, a ****iter*** vraća kao rezultat element u kontejneru na koji pokazuje ***iter***. Za svaki tip kontejnera možemo reći da podržava sledeće funkcije:

begin() – funkcija koja kao rezultat vraća iterator koji pokazuje na prvi element u kontejneru

end() – funkcija koja kao rezultat vraća iterator koji pokazuje na element za 1 iza poslednjeg elementa.

Iterator je ime tipa koji je definisan u kontejneru. Za klasu vector sintaksa je:

Vector<string> :: iterator

Osim tog iteratora, u okviru svakog kontejnera je definisan i tip *const iterator* koji je neophodan za pristupanje elementima konstantnih kontejnera. Ovakav tip iteratora dozvoljava samo čitanje elemenata kontejnera[12].

3.5.2. Vektor (vector)

Pod vektorom podrazumevamo susedne oblasti memorije u kojima se svaki element čuva određenim redosledom. Njegove osnovne karakteristike su:

1. Efikasni nasumični pristup elementima vektora (predstavlja fiksni pomeraj u odnosu na početak vektora).
2. Umetanje elemenata bilo gde osim na kraj vektora nije efikasno, jer bi zahtevalo premeštanje svih elemenata udesno od umetnutog za jedno mesto udesno.
3. Brisanje elemenata sa bilo kog mesta osim sa kraja nije efikasno jer bi zahtevalo premeštanje svih elemenata desno od izbrisanog za jedno mesto ulevo[12].

3.5.3. Dinamika vektora

Da bi vektor dimanimički rastao, on mora da obezbedi dodatnu memoriju za čuvanje nove sekvence, da redom iskopira elemente stare sekvence i da oslobodi memoriju koju je zauzimala stara sekvenca. Ako bi se vektor povećavao posle svakog umetanja to bi bilo neefikasno, pogotovu ako su elementi vektora objekti klasa, pa je pri kopiranju za svaki element potrebno pozvati konstruktor kopije a pri brisanju destruktor za taj element[12].
Zato, kada se javi potreba za povećanjem vektora, obezbeđuje se dodatni kapacitet memorije koji prevazilazi trenutne potrebe vektora i čuva se u rezervi[12].

3.5.4. Liste (list)

Lista predstavlja nesusedne oblasti memorije koje su dvostruko povezane parom pokazivača koji pokazuju na prethodni i sledeći element omogućavajući pri tome istovremeno pristupanje elementima i unapred i unazad. Osnovne karakteristike liste su:

1. Nasumični pristup elementima liste nije efikasan. Naime, da bi se pristupilo nekom elementu neophodno je pristupiti svim prethodnim elementima.
2. Umetanje i brisanje elemenata na bilo kom mestu u listi je efikasno. Potrebno je samo premestiti pokazivače ali elemente nije potrebno dirati.
3. Potrebna je dodatna memorija za po dva pokazivača za svaki element[12].

3.5.5. Dek (deque)

Kontejner koji je optimizovan za dodavanje/brisanje elemenata sa oba kraja (sa početka i sa kraja). Za raliku od vektora, ne čuva elemente u uzastopnim memoriskim lokacijama, već u blokovima uzastopnih memorijskih lokacija. Zbog toga nema realokacije memorije prilikom dodavanja elemenata, što je znatna prednost u odnosu na vektor – kada je potrebno proširiti dek, alocira se novi blok memorije za nove elemente. Omogućava indeksni pristup (ima operator[]), ali nešto sporiji od vektora.

3.5.6. Skup (set) i katalog (map)

Predstavljaju osnovne tipove asocijativnih kontejnera. Kontejner skup se sastoji iz ključnih vrednosti i vrši efikasnu proveru prisustva nekog elementa. Kod kontejnera katalog imamo elemente koji predstavljaju parove koji su sastavljeni od ključa i vrednosti. Ključ se koristi za pretraživanje, a vrednost sadrži neki podatak koji želimo da koristimo. Primer za to je telefonski imenik, gde je ime pojedinca ključ, dok je njegov telefonski broj vrednost.[12]

Uz sve do sada navedene klase kontejnera treba dodati da se u standardnoj biblioteci u kojoj su definisane navedeni kontejneri, nalazi i mnoštvo operacija (funkcija) koje se mogu primeniti na njima. Ove operacije nisu obezbeđene kao funkcije šabloni navedenih klasa, već kao nezavisni skup generičkih algoritama u koje ubrajamo jedan manji skup, a to su:

- Algoritam pretraživanja,
- Algoritmi sortiranja,
- Numerički algoritmi,i dr.

Da bi se odgovarajući algoritmi primenili u programu, neophodno je pozvati odgovarajuće zaglavlje.[12]

4. OBJEKTNO ORIJENTISANA SIMULACIJA JEDINICE POŠTANSKE MREŽE

Objektno orijentisana simulacija (OOS) koju smo primenili u rešavanja problema vezanog za kvalitet opsluge korisnika na šalterima pošte prate definicije klasa, struktura, funkcija, promenljivih koje opisujemo u programskom jeziku C++.

Važno je napomenuti da se može zapaziti poistovećivanje elemenata programskog koda u C++ jezika i elemenata i naredbi koje se koriste u GPSS-u, kada se vrši uporedna analiza ova dva jezika. Svaka od narednih klasa sa svojim metodama može da nađe sličnu naredbu u GPSS – u. Kada u izvornom kodu u jeziku C++ uz objekat određene klase postavimo neku od metoda klase, onda takvom linijom koda dobijamo naredbu u GPSS jeziku.

U radu se za realizaciju OOS koristi zaglavlje "simul.h" koji predstavlja deo programskog koda u kome se definišu neophodne klase i to su:

Klasa „raspodela" u kojoj mogu biti definisane različite metode i promenljive koje opisuju određeni tip raspodele, poput normalne, uniformne, eksponencijalne itd. U pomenutoj klasi se nalazi metoda kojom se generišu slučajni brojevi koji se koriste u izvornom kodu programa. Ova metoda potpuno odgovara generatoru slučajnih brojeva koje koristimo u GPSS-u (RNj).

Klasa "entitet",entiteti C++ jezika u GPSS-u jeziku bi predstavljale transakcije koje se kreću kroz blok naredbe. Ovom klasom se opisuju redni broj transakcije, vreme u koje nastupa transakcija, parametri transakcije

Klasa "statistike" definiše broj entiteta koji prođe kroz simulaciju, maksimalni i trenutni broj entiteta, vremena entiteta koji nisu čekali, vremena zadržavanja entiteta u resursu, ukupno vreme zauzetosti resursa (šaltera). Na ovaj način definisana klasa treba da predstavlja blokove GPSS – a kojim se mogu tabelirati vrednosti dužine reda čekanja, vremena provedenog u redu itd.

Klasa "simulacija" je klasa kojom se omogućava izvršavanje programa putem metoda koje su definisane u njoj. Ona kreira u uništava entitete, raspoređuje ih na određeni događaja koji predstoji određenom entitetu. Praktično klasa "simulacija" ne odgovara nijednom od blokova u GPSS-u, već ona odgovara samom toku kretanja transakcija.

Klasom "red_cekanja" entitete smeštamo u redove određenim šaltera iz kojih ih vadimo kada je predviđeni šalter slobodan ili kada je vreme za njihovo nastupanje na redu. Blok QUEUE, GPSS jezika, je blok koji se uvodi u model radi prikupljanja statistike o čekanju transakcije na nekom od blokova. Ovaj blok možemo da poistovetimo sa klasom"red_čekanja" kada uz objekat klase pozovemo metodu *ured(e),* pri čemu entiteti koji izlaze iz reda čekanja predstavljaju transakciju u bloku QUEUE. Blok DEPART bi se onda ponašao kao linija koda u C++ *red_uni.izred().*

Poslednja definisana klasa zaglavlja "simul.h" jeste klasa "resurs", koja predstavalja šalter na kom se opslužuju korisnici (entiteti), koje se izvode iz reda čekanja, preusmeravaju se na predviđeni šalter i opslužuju. Po njihovom opsluživanju oni napuštaju šalter, odnosno jedinicu poštanske mreže. Analogno ovoj klasi, u GPSS imamo blokove za zauzimanje uređaja ili skladišta i za oslobađanje šaltera (skladišta), SEIZE (ENTER), RELEASE (LEAVE).

Vremena dolazaka korisnika, verovatnoće izbora određene usluge, kao i vremena opsluge na šalterima smo generisali pomoću dve funkcije koje su opisane u jednoj strukturnoj celini. Jedna od njih jeste diskretna funkcija koja vraća vrstu usluge koju je korisnik odabrao. Takvoj funkciji se prosleđuju sledeći parametri:

- objekata klase raspodela koji se pomoću operatorske funkcije "zagrada" () vraća realna vrednost u granicama od nule do jedan.
- realan niz koji zapravo predstavlja kumulativnu frekvenciju izbora vrste usluge.
- realan niz, iz kog svaki broj predstavlja konkretnu uslugu
- broj tačaka navedenih realnih nizova.

U zavisnosti od izabrane usluge razlikovaćemo vremena opsluge na šalterima koja su data kao srednja vrednost za svaku od usluga.

Kontinualna funkcija predstavlja nešto slično diskretnoj pri čemu naglašavamo, da se vrši linearna interpolacija unutar članova nizova kojima se vraća realno vreme dolazaka korisnika.

5. KARAKTERISTIKE ŠALTERSKE SLUŽBE POŠTE 11158 BEOGRAD

Javni poštanski operatori kao pružaoci univerzalne poštanske usluge imaju obavezu da obezbede teritorjalnu, vremensku, personalnu i finansijsku dostupnost svojih usluga. To podrazumeva obavezu da poštanskom mrežom bude obuhvaćeno svako naseljeno mesto u zemlji, da u svakom od njih bude formirana jedinica poštanske mreže u kojoj će biti organizovan prijem, prevoz i distribucija pošiljaka, kao i vrešenje drugih usluga za potrebe korisnika.

Vremenska dostupnost podrazumeva to da poštanske usluge moraju biti dostupne korisnicima svakog radnog dana, a ne manje od pet dana u nedelji osim za vreme državnih i verskih praznika. Takođe, mora biti utvrđeno radno vreme koje će odgovarati korisnicima, ali i u skladu sa rangom jedinice poštanske mreže i zahtevima za uslugama u neseljenom mestu u kome je organizovana. Stalne pošte su u obavezi da rade najmanje dva sata dnevno.

Pod personalnom dostupnošću podrazumeva se pravo svakog pravnog ili fizičkog lica da neograničeno potražuje poštanske usluge. Ukoliko prethodno ispuni propisane uslove (dimenzije, pakovanje, plaćanje poštarine itd.) klijent može svaku pošiljku predati u svakoj jedinici poštanske mreže za pružanje usluga korisnicima i slobodno birati određeni vid usluđivanja[13].

Jedinica poštanske mreže 11158 Beograd 118 nalazi se u Dubrovačkoj Ulici br.35, na teritoriji opštine Stari grad Pružanje usluga korisnicima u pošti 11158 Beograd118 organizovano je na pet šaltera, šest dana u nedelji i to radnim danima u dve smene a subotom samo u prepodnevnoj smeni. U ovoj pošti postoje tri šaltera za novčano poslovanje, jedan šalter za pismonosne i paketske usluge i peti šalter koji služi za isporuku pošiljaka. Za svakodnevne potrebe korisnika u pošti Beograd 118 aktivna su dva šaltera, jedan za novčano poslovanje i drugi za pismonosne i paketske usluge.

Na šalteru za novčano poslovanje pružaju se sledeće usluge:

- naplata telefonskih računa, električne energije i komunalnih usluga
- uplata i isplata sa tekućih računa Banke Poštanske štedionice
- uplata i isplata dinarske štednje kod Banke Poštanske štedionice
- isplata čekova po tekućim računima
- isplata sa računa građana na osnovu platnih kartica drugih banaka
- uplata na žiro račune pravnih lica
- uplata i isplata PostNet i Western Union uputnica

Na šalteru pismonosnoh i paketskih usluga se mogu vršiti sledeće usluge:

- prijem običnih pismonosnih pošiljaka
- prijem preporučenih pošiljaka
- prijem vrednosnih pošiljaka
- prodaja poštanskih vrednosti
- prijem pojedinačnih paketa
- prijem paketa na osnovu prijemne knjige-lista
- prijem Post Express pošiljaka

U programskom delu završnog rada biće realizovana objektno orijentisana simulacija pošte 118 na šalteru novačnog poslovanja i šalteru za pismonosne i paketske usluge. Takođe, biće ispitivani parametri koji su vezani za kvalitet opsluge korisnika na pomenutim šalterima.

6. PRIKUPLJANJE I STATISTIČKA ANALIZA PODATAKA ŠALTERSKE SLUŽBE KAO SISTEMA MASOVNOG OPSLUŽIVANJA

Statistika kvantitativno istražuje pojave koje su po svojoj prirodi varijabilne i imaju masovni karakter. Ponašanje tih pojava nije unapred određeno, pa se statističkom analizom otkrivaju i utvrđuju zakonitosti koje u njima vladaju. Skup svih elemenata na osnovu koji se određena pojava statistički posmatra zove se statistički skup odnosno populacija. Pojedinačni elementi iz kojih se statistički skup sastoji zovu se elementi statističkog skupa, odnosno statističke jedinice.[11]

Definisanje statističkog skupa zavisi od prirodne pojave koja se želi analizirati, od cilja istraživanja i raspoloživih mogućnosti posmatranja. Mora se voditi računa o tome da skup bude homogen, odnosno da su statističke jedinice posmatrane populacije u suštini slične i pokazuju razlike samo u pogledu obeležja koja se ispitiju. Što statističke jedinice imaju više zajedničkih osobina, to je statistički skup homogeniji. Ispitivanje i određivanje varijacija koje se javljaju u pogledu nekih obeležja statističkih jedinica jeste cilj statističkih analiza. Uglavnom je nemoguće i neopravdano vršiti statističku analizu na celom statističkom skupu. Zbog toga se vrlo često vrši odabir nekih elemenata populacije na kojima se sprovodi dalja statistička analiza. Načini na koje se vrši odabir elemenata nisu proizvoljni već moraju ispuniti određene zahteve, a u cilju da njihov skup, odnosno uzorak na kome se vrši analiza bude reprezentativan. To podrazumeva da uzorak mora verno odražavati strukturu populacije koju predstavlja i da zakonitosti koje vladaju u njemu mogu biti generalizovane za celu populaciju kojoj pripada[11].

Realni sistem je izvor podataka za prikupljanje relevantnih podataka o aktivnostima koje se u njemu dešavaju, kao i za definisanje modela na osnovu koga će se vršiti analiza njegovog funkcionisanja. Osnovni zahtev koji se postavlja prilikom prikupljanja podataka jeste njihova tačnost, istinitost i potpunost. Da bi se izvela uspešna analiza sistema metodama simulacije, potrebno je formirati empirijske raspodele dolazaka i opsluge klijenata snimanjem saobraćaja u konkretnoj pošti na koju se analiza odnosi. Prikupljanje podataka, odnosno snimanje saobraćaja, mora se izvoditi planski i sistematski u tačno određenom periodu u toku dana, meseca i godine da bi se obuhvatile dnevne, mesečne i sezonske oscilacije saobraćaja[11].

Treba napomenuti da se za potrebe simulacije i teorije masovnog opsluživanja pri analizi funkcionisanja poštanske šalterske službe mogu koristiti i teorijske i empirijske funkcije raspodele ulaznih parametara. Ako se prilikom analize sistema koriste teorijske raspodele, najpre se mora testirati slaganje ulaznih podataka sa zakonitostima koje vladaju u njima. U ovom radu će biti korišćene empirijske raspodele ulaznog toka klijenata i vremena opsluživanja, bez izjednačavanja i zamenjivanja teorijskim[11].

6.1. Podaci o pristupanju klijenata šalteru uplate – isplate i šalteru za pismonosne i paketske usluge

Dolasci klijenata na šaltere novčanog poslovanja mereni su u periodu od po sat vremena u toku pet radnih dana. Beleženi su vremenski periodi između nailazaka dva uzastopna korisnika. U tabelama 6.1 i 6.2 se mogu videti u kom procentu i u kom periodu su korisnici pristupali pošti.

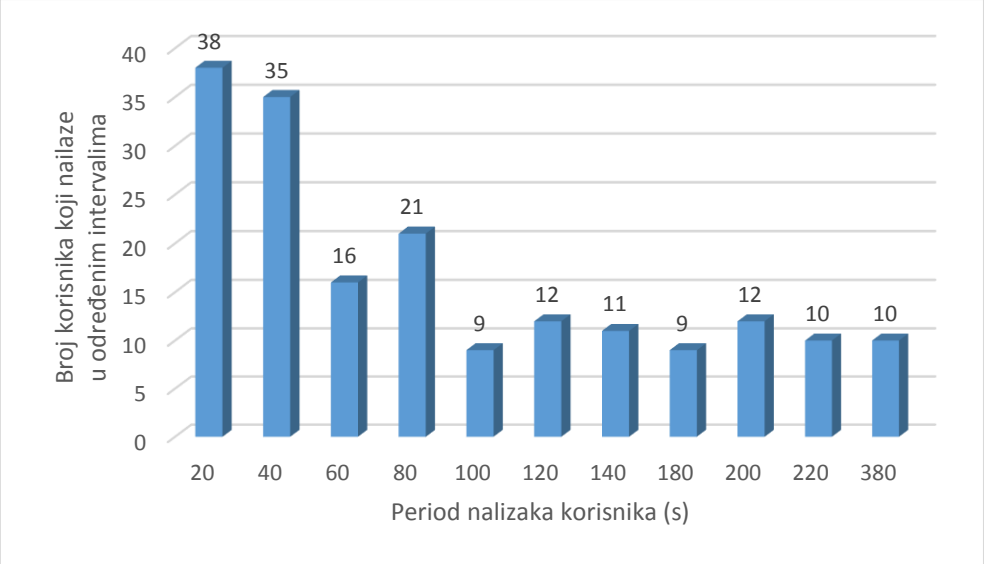
Tabela 6.1. Statistika dolazaka korisnika na šalter novčanog poslovanja

Klase (a_i, b_j]	Frekvencije	Relativne verovatnoće	Kumulativne verovatnoće
0	0	0,000	0,000
0-20	38	0,208	0,208
20-40	35	0,191	0,399
40-60	16	0,087	0,486
60-80	21	0,115	0,601
80-100	9	0,049	0,650
100-120	12	0,065	0,715
120-140	11	0,060	0,775
140-180	9	0,049	0,824
180-200	12	0,065	0,890
200-220	10	0,055	0,945
220-380	10	0,055	1,000
∑	183	1,000	

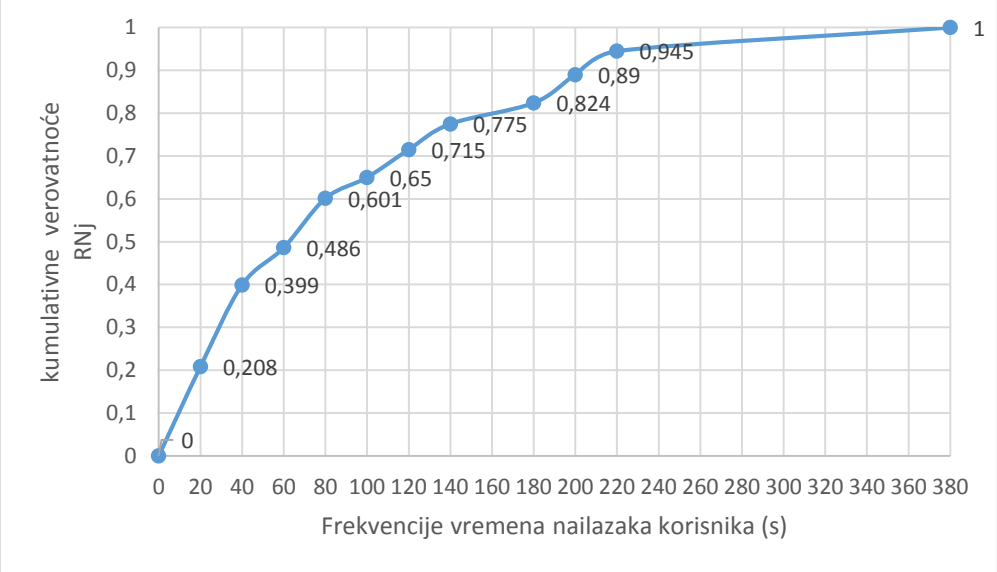
Tabela 6.2. Statistika dolazaka korisnika na šalter poštanskih pošiljaka

Klase (a_i, b_j]	Frekvencije	Relativne verovatnoće	Kumulativne Verovatnoće
0	0	0,000	0,000
0-200	15	0,417	0,417
200-400	11	0,305	0,722
400-800	5	0,139	0,861
800-1000	5	0,139	1,000
∑	36	1,000	

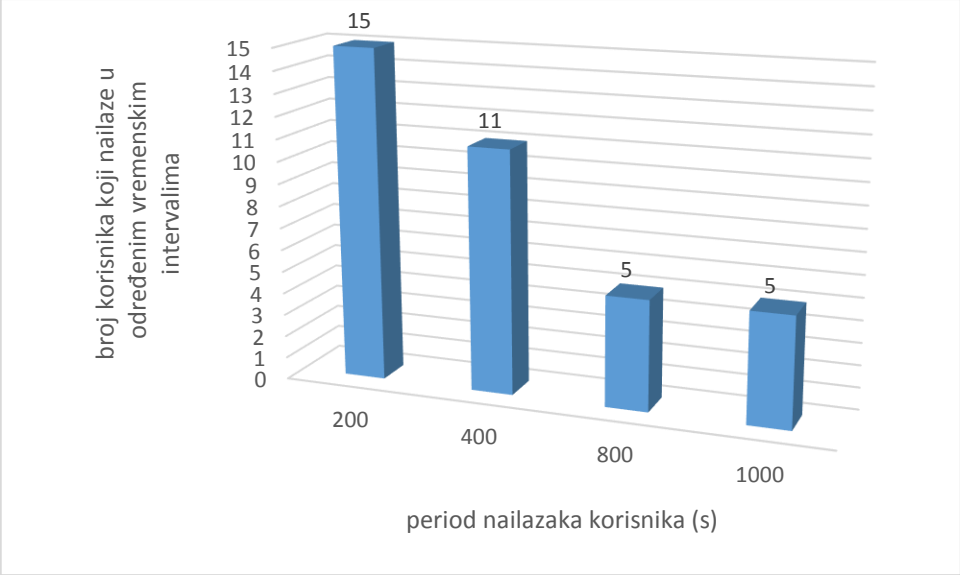
Na slikama 6.1 i 6.3 su prikazani histogrami raspodele frekvencija vremena nailazaka korisnika koji su konstruisani na osnovu podataka dobijenih merenjem vremena između njihovih nailazaka. Na njima se mogu videti broj korisnika koji su u toku zadatok intervala pristupili sistemu posle prethodno prispelih korisnika. Na osnovu histograma sa formiraju empirijske funkcije raspodela vremena između dolazaka prikazane na slikama 6.2 i 6.4.



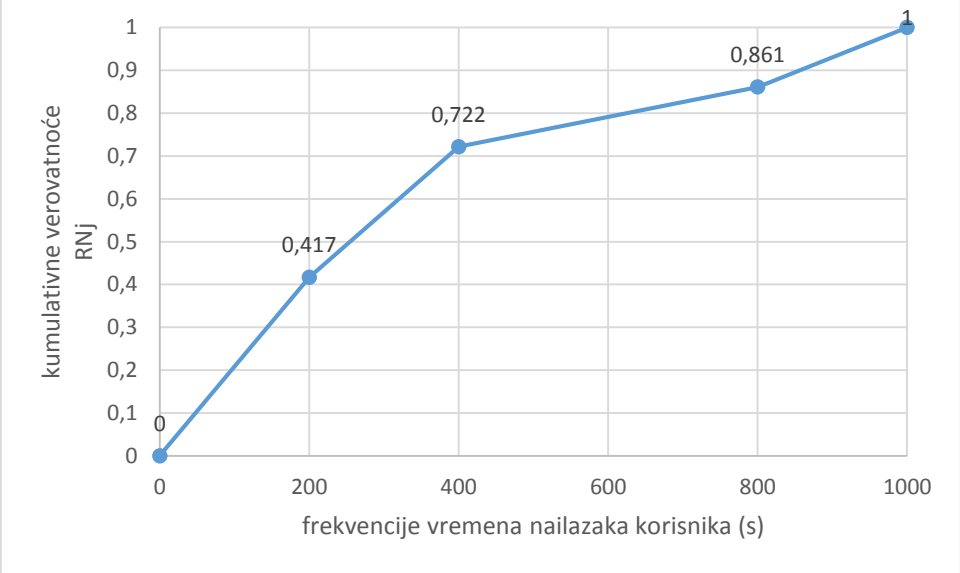
Slika 6.1.Histogram raspodele frekvencija vremena nailazaka korisnika na šalter uplate – isplate



Slika 6.2. Kumulativna empirijska funkcija raspodele vremena nailazaka korisnika na šalter uplate – isplate

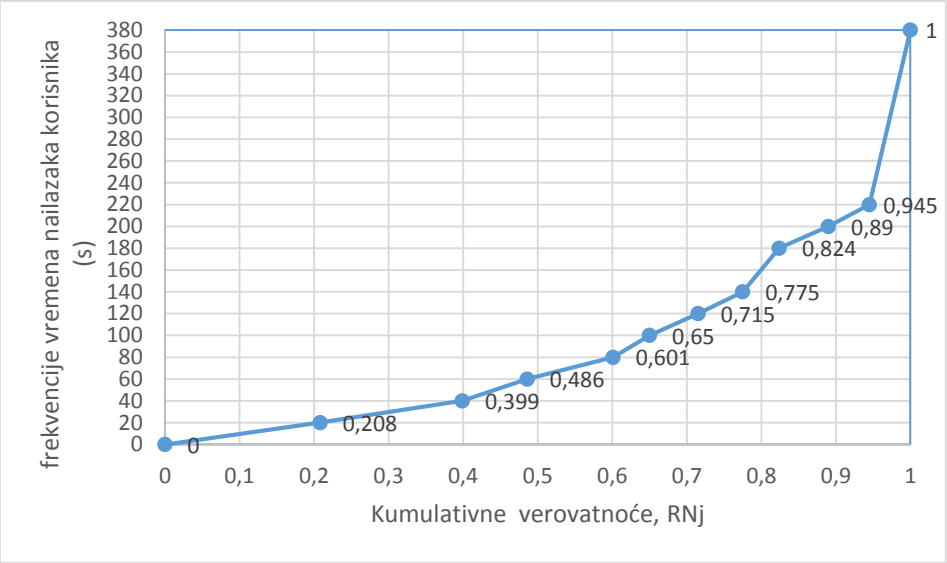


Slika 6.3. Histogram raspodele frekvencija vremena nailazaka korisnika na pismonosni šalter

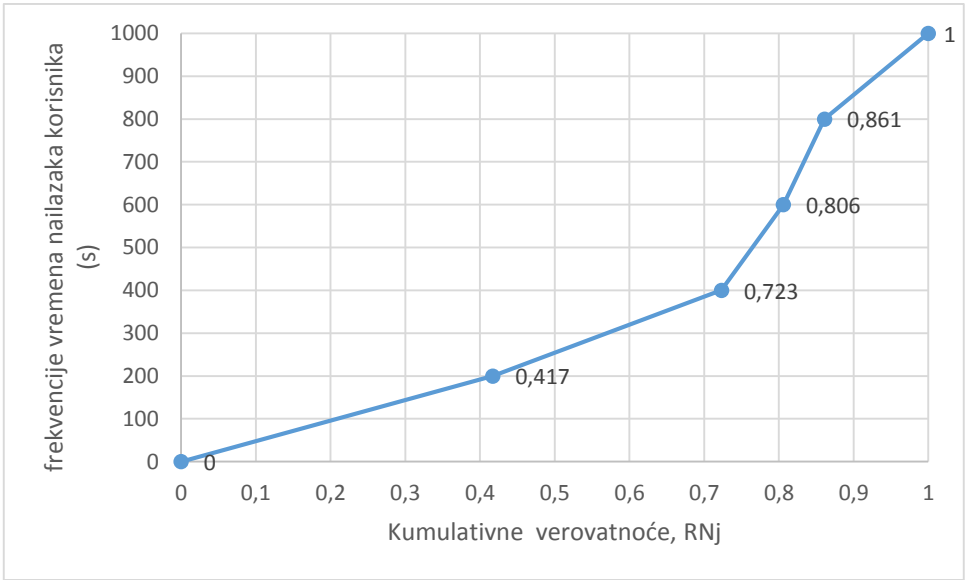


Slika 6.4.Kumulativna empirijska funkcija raspodele vremena nailazaka korisnika za pismonosni šalter

Za potrebe simulacije, odnosno za definisanje funkcija koje se direktno koriste prilikom simulacije sistema, koriste se inverzne kumulativne empirijske funkcije raspodela za oba šaltera i biće prikazane na slikama 6.5 i 6.6. Na njima se može videti koja je verovatnoća da se vrednost funkcije nađe u bilo kom intervalu na osnovu podataka iz tabela 6.1 i 6.2.



Slika 6.5. Inverzna kumulativna empirijska funkcija raspodele vremena nailazaka korisnika na šalter uplate – isplate



Slika 6.6. Inverzna kumulativna empirijska funkcija raspodele vremena nailazaka korisnika na pismonosni šalter

6.2.Podaci o opsluživanju korisnika na šalteru novčanog poslovanja

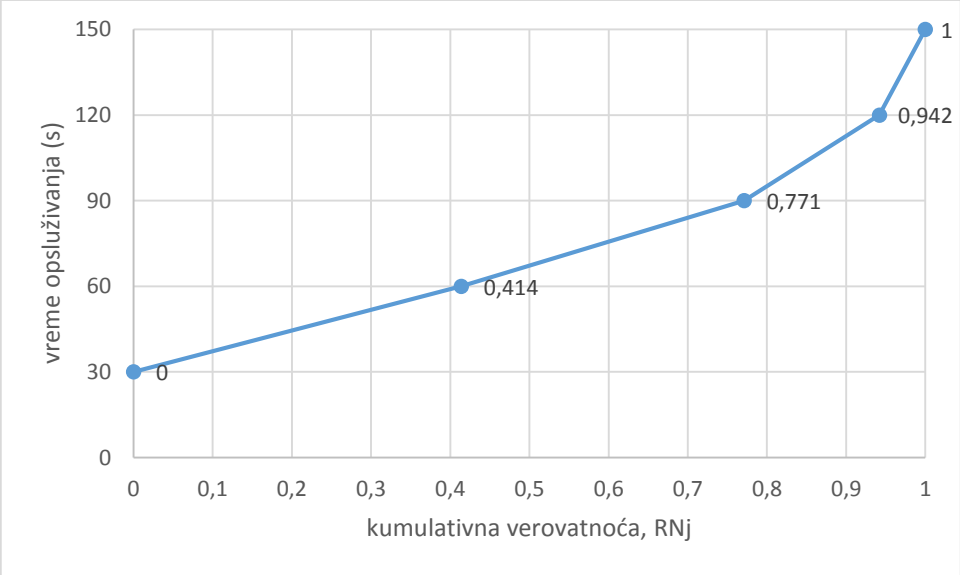
Podaci o opsluživanju korisnika su dobijeni na osnovu merenja trajanja opsluge na šalterima uplate – isplate (novčano poslovanje) i na šalteru za pismonosne i paketske usluge (pismonosni šalter). Za svaki od navedenih šaltera zapazili smo nekoliko najčešće traženih usluga i to, pet usluga za novčano poslovanje i pet pismonosno – paketskih usluga. U nastavku predstavljamo tabelarno i grafički svaku od deset usluga, tabelama 6.3 - 6.7 i slikama 6.7 - 6.11.

U grafičkom prikazu su nam nephodne inverzne empirijske funkcije raspodele vremena opsluživanja za svaku od usluga. Na osnovu njih će biti definisane raspodele trajanja opsluge u simulacionom modelu, odnosno vreme zadržavanja transakcija u toku prolaska kroz model.

6.2.1. Nalog za uplatu

Tabela 6.3. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *nalog za uplatu*.

Vreme opsluge(s)	30,00	60,00	90,00	120,00	150,00
Kumulativna raspodela verovatnoća	0,000	0,414	0,771	0,942	1,000

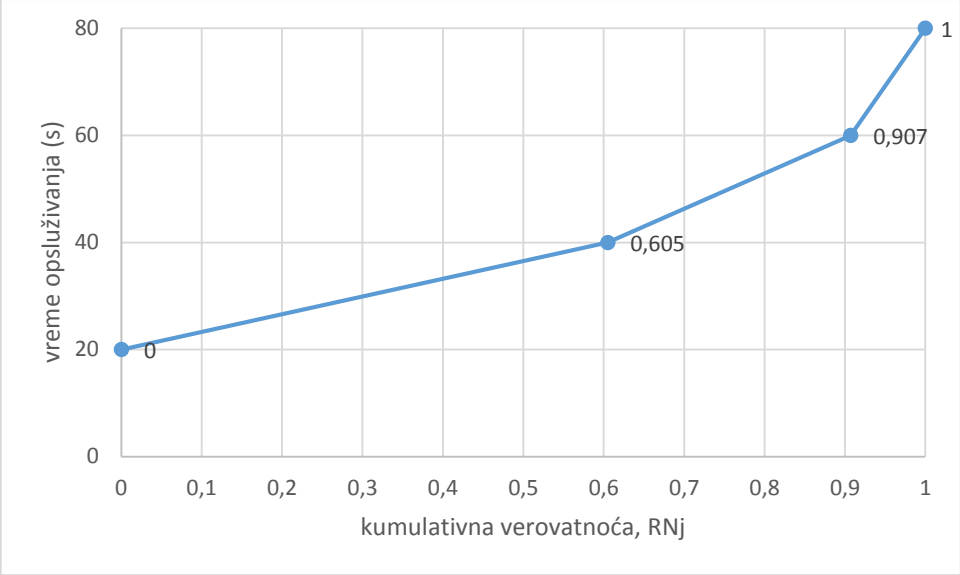


Slika 6.7.Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za usluge *nalog za uplatu.*

6.2.2. Uplate telefonskih računa

Tabela 6.4. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *uplate telefonskih računa.*

Vreme opsluge(s)	20,00	40,00	60,00	80,00
Kumulativna raspodela verovatnoća	0,000	0,605	0,907	1,000

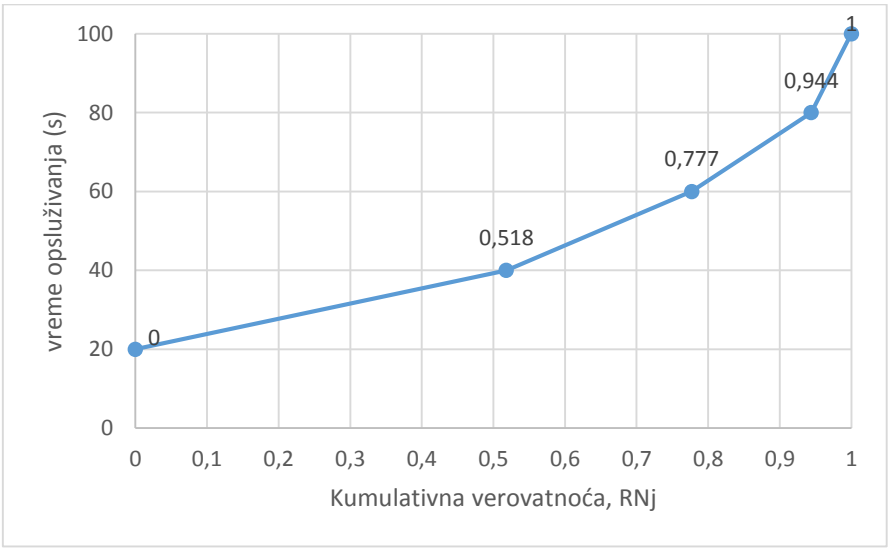


Slika 6.8. Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za usluge *uplate telefonskih računa.*

6.2.3. Uplate komunalnih usluga

Tabela 6.5. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *uplate komunalnih usluga.*

Vreme opsluge(s)	20,00	40,00	60,00	80,00	100,00
Kumulativna raspodela verovatnoća	0,000	0,518	0,777	0,944	1,000

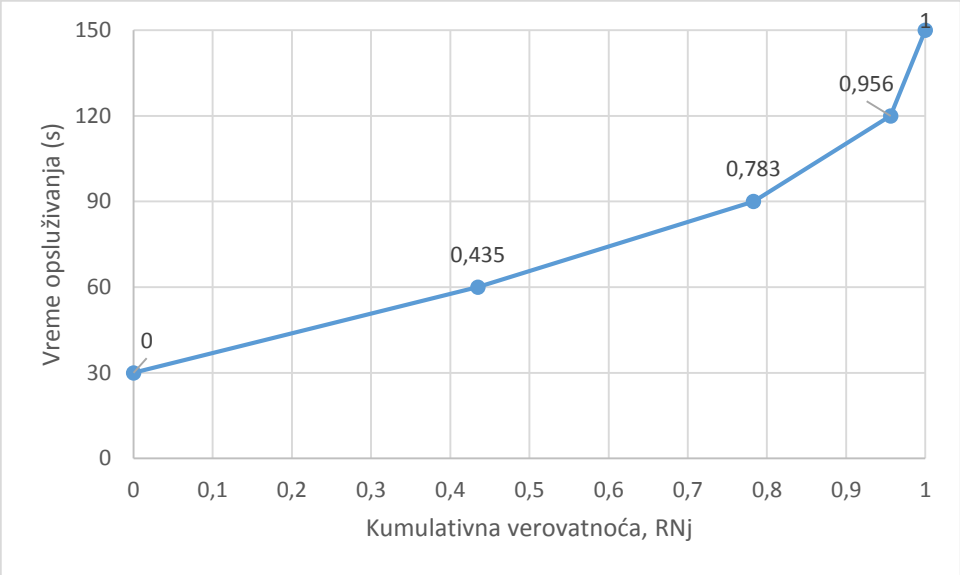


Slika 6.9. Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za usluge *uplate komunalnih usluga*

6.2.4. Isplate na POS terminalima

Tabela 6.6. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *isplate na POS terminalima.*

Vreme opsluge(s)	30,00	60,00	90,00	120,00	150,00
Kumulativna raspodela verovatnoća	0,000	0,435	0,783	0,956	1,000

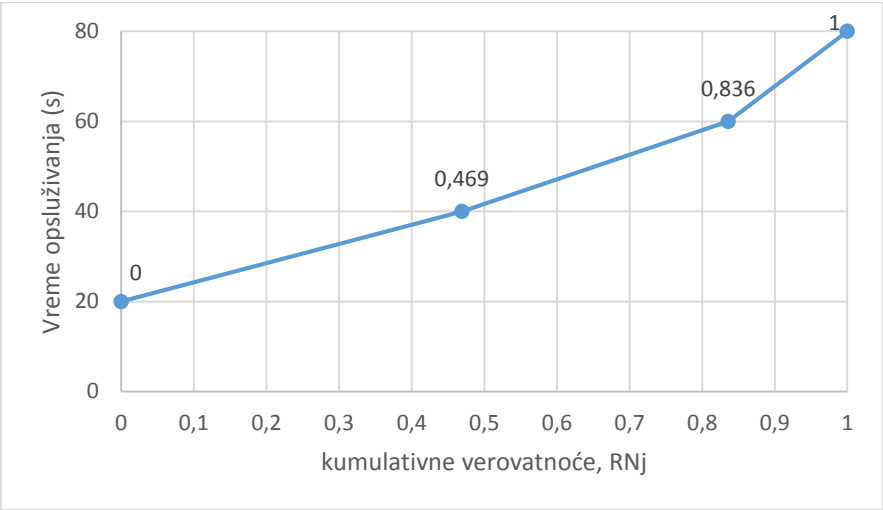


Slika 6.10. Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za usluge *isplate na POS terminalima*

6.2.5. Uplate računa za infostan

Tabela 6.7. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *uplate računa za infostan.*

Vreme opsluge(s)	20,00	40,00	60,00	80,00
Kumulativna raspodela verovatnoća	0,000	0,469	0,836	1,000



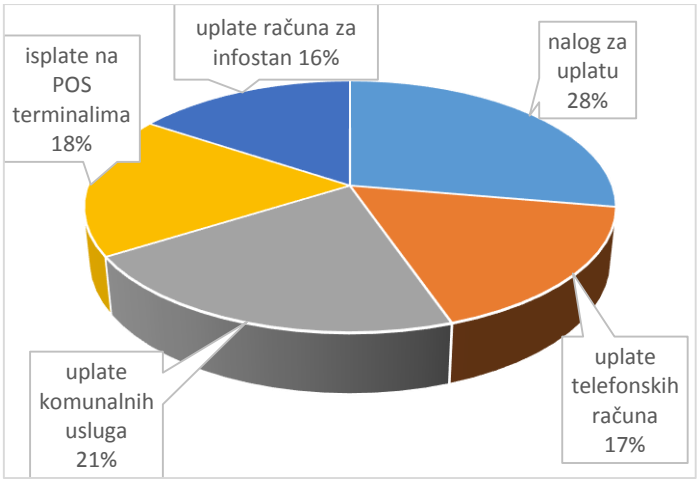
Slika 6.11.Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za usluge *uplate računa za infostan*

6.3.Podaci o verovatnoći izbora usluge

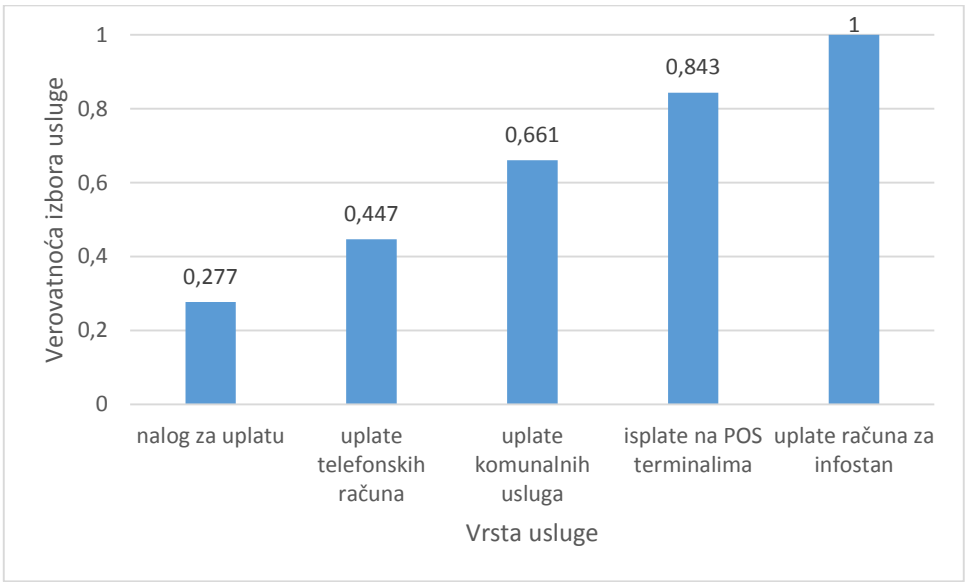
Verovatnoća izbora usluge određuje se izračunavanjem zastupljenosti pojedinih usluga koje su se vršile za vreme snimanja vremena opsluživanja. Na slikama 6.12 i 6.14 su grafički prikazane procentualne zastupljenosti usluga novčanog poslovanja. Na osnovu podataka o broju transakcija svake usluge pravimo i kumulativne funkcije raspodele verovatnoća izbora usluge, koje su prikazane tabelom 6.8 i slikom 6.13. Pored podataka o procentualnoj zastupljenosti svake usluge, možemo da zapazimo da korisnik koji se opslužuje može zahtevati više različitih usluga u toku svog vremena opsluživanja i od takve pojave možemo generisati podatke koji bi bili važni za funkcionisanje modela. Za naš model, naglašavamo da korisnik može da zahteva jednu ili dve usluge u toku svog vremena opsluživanja.

Tabela 6.8. Kumulativna empirijska funkcija raspodele verovatnoća odabira vrste usluge

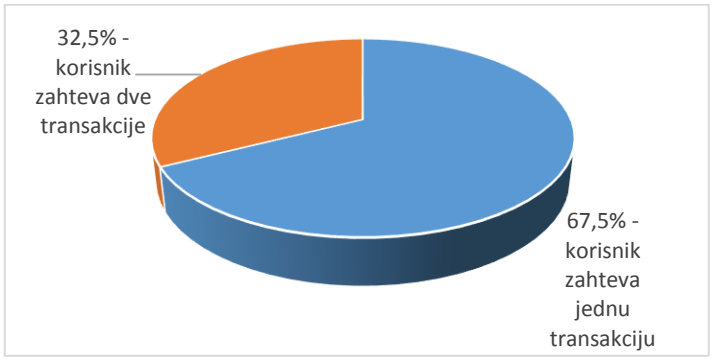
vrsta usluge	Nalog za uplatu	Uplate telefonskih računa	Uplate komunalnih usluga	Isplate na POS terminalima	Uplate računa za infostan
relativne verovatnoće	0,277	0,170	0,214	0,182	0,157
Kumulativne verovatnoće	0,277	0,447	0,661	0,843	1,000



Slika 6.12. Procentualna zastupljenost novčanih usluga na šalterima novčanog poslovanja



Slika 6.13.Kumulativna funkcija raspodele verovatnoća izbora usluge



Slika 6.14. Procentualni odabir broja transakcija koje korisnik zahteva

6.4.Podaci o opsluživanju korisnika na šalteru poštanskih pošiljaka

Posmatranjem liste transakcija na šalterima za prijem poštanskih pošiljaka, uočeno je pet usluga za koje se najčešće vrši prijem:

- pismo u unutrašnjem poštanskom saobraćaju,
- postexpress pošiljaka,
- preporučena pošiljaka (opremljeni popis),
- obično pismo – grupni prijem
- paket u unutrašnjem poštanskom saobraćaju.

Pod prijemom pisama u unutrašnjem poštanskom saobraćaju (UPS – u) smatraju se sledeće usluge:

- prijem pisma putem terminala (bez posebnih usluga),
- prijem pošiljaka sa posebnim uslugama (preporučena pošiljka, pošiljka sa povratnicom)

Postexpress pošiljke koje se primaju na šalterima pošta su one pošiljke koje se uručuju narednog dana primaocima na kućnoj adresi do 12 ili 19 časova (usluga „Danas za sutra“).
Korisnici koji redovno ili povremeno šalju određene količine pošiljaka na različite adrese, sklapaju ugovor sa preduzećem „Pošta Srbije“ kojim se definišu cene odgovarajućih usluga (poštarine) i način na koji će oni vršiti predaju pošiljaka. Predaja pošiljaka se može izvršiti na dva načina:

- putem neopremljenog popisa i
- putem opremljenog popisa.

Korisnici koji su se odlučili da pošiljke predaju putem opremljenog popisa od pošte dobijaju određenu količinu nalepnica na kojima se nalaze alfanumeričke oznake, obrazac „Prijemna knjiga – list“ i cenovnik poštanskih usluga. Korisnici se izborom ove usluge obavezuju da će pošiljke „opremiti“ i takve ih predati na prenos.
Kada pošiljalac šalje preporučene pošiljke putem opremljenog popisa, on je dužan da na svaku pošiljku stavi nalepnicu koja predstavlja prijemni broj pošiljke, da pošiljke redom upiše u Prijemnu knjigu prema prijemnim brojevima (u rastućem nizu) i da za svaku pošiljku u zavisnosti od mase i posebne usluge (povratnica) obračuna poštarinu i iznos poštarine upiše u Prijemnu knjigu. Pošiljalac je takođe u obavezi da sortira pošiljke saglasno prijemnim brojevima pošiljaka.Korist od ove usluge je obostrana, jer se šalterskom radniku olakšava posao na prijemu, a pošiljalac dobija određeni popust.Korisnici koji predaju veće količine pošiljaka bez posebnih usluga takođe mogu da sklope ugovor sa poštom, i u tom slučaju se u manipulativnim ispravama unosi ukupan broj takvih pošiljaka, a takođe se vrši obrada putem terminala. U ovu grupu spadaju i pošiljke direktne pošte.

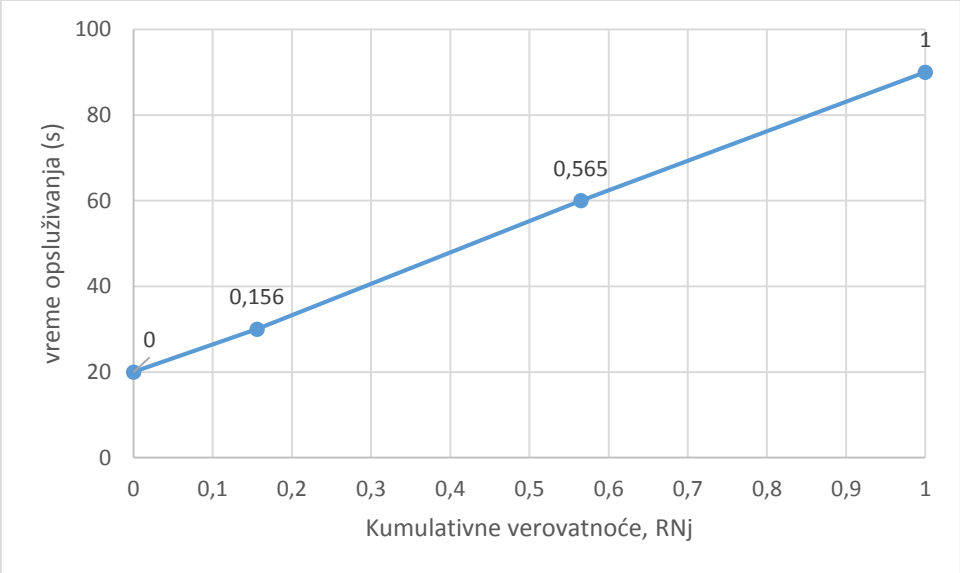
Paketske usluge su usluge prenosa zatvorenih pošiljaka (paketa), sa označenom vrednošću i registrovanim brojem prijema koje, po pravilu, sadrže robu i druge predmete.

Na šalteru za pismonone i paketske usluge razlikujemo pet usluga koje navodimo u radu. S obzirom da smo u programskom delu završnog rada u delu koji opisuje pismonosno-paketski šalter naglasili da su sve usluge osim usluge *pismo UPS* imaju diskretnu raspodelu verovatnoća, iz tog razloga prikazujemo grafički samo uslugu *pismo UPS,* koja ima kontinualnu karakteristiku. Tabelama 6.9 – 6.14 su predstavljene kumulativne empirijske funkcije raspodele vremena i verovatnoće izbora svake usluge, dok su slikam 6.15 i 6.16 prikazane inverzne kumulativne empirijske funkcije i slikom 6.17 procentualna zastupljenost usluga, pojedinačno, na šalteru za prijem paketa i pošiljaka.

6.4.1. Pismo UPS

Tabela 6.9. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *pismo UPS*

Vreme opsluge(s)	20	30	60	90
Kumulativna raspodela verovatnoća	0,000	0,156	0,565	1,000



Slika 6.15. Inverzna kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *pismo UPS*

6.4.2. Post Ekspres

Tabela 6.10. kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *Post Ekspres*

Vreme opsluge(s)	120	180	240
Kumulativna raspodela verovatnoća	0,500	0,833	1,000

6.4.3. R – opremljen popis

Tabela 6.11. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *R-opremljen popis*

Vreme opsluge(s)	30	60	120
Kumulativna raspodela verovatnoća	0,200	0,900	1,000

6.4.4. O – pošiljke grupno

Tabela 6.12. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *O-pismo grupno*

Vreme opsluge(s)	60	120
Kumulativna raspodela vetovatnoća	0,571	1,000

6.4.5. Paket UPS

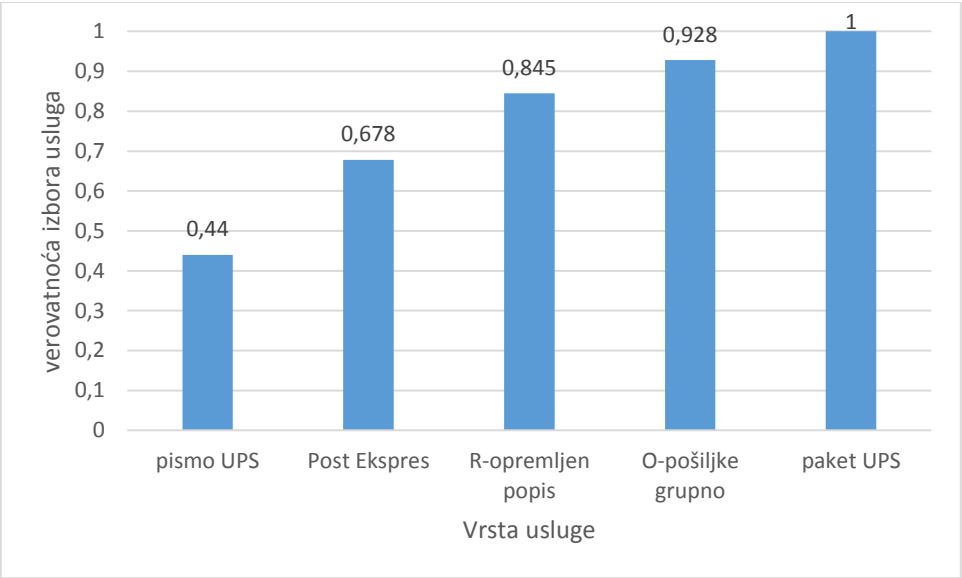
Tabela 6.13. Kumulativna empirijska funkcija raspodele vremena opsluge za uslugu *paket UPS*
grupno

Vreme opsluge(s)	90	120
Kumulativna raspodela vetovatnoća	0,167	1,000

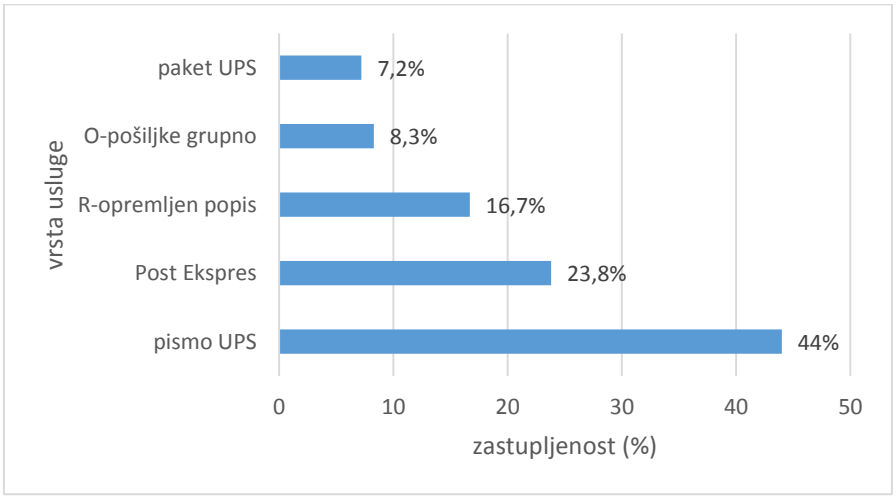
6.4.6. Verovatnoće izbora usluga

Tabela 6.14. Kumulativna empirijska funkcija raspodele verovatnoća odabira vrste usluge

vrsta usluge	Pismo UPS	Post Ekspres	R-opremljen popis	O-pošiljke grupno	Paket UPS
relativne verovatnoće	0,440	0,238	0,167	0,083	0,072
Kumulativne verovatnoće	0,440	0,678	0,845	0,928	1,000



Slika 6.16. Kumulativna funkcija raspodele verovatnoća izbora usluge



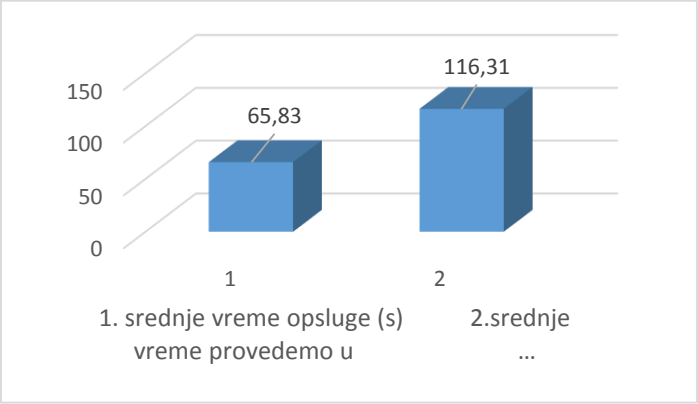
Slika 6.17.Procentualna zastupljenost usluga na pismonosno-paketskom šalteru

6.5.Postojeće stanje organizacije šalterske službe

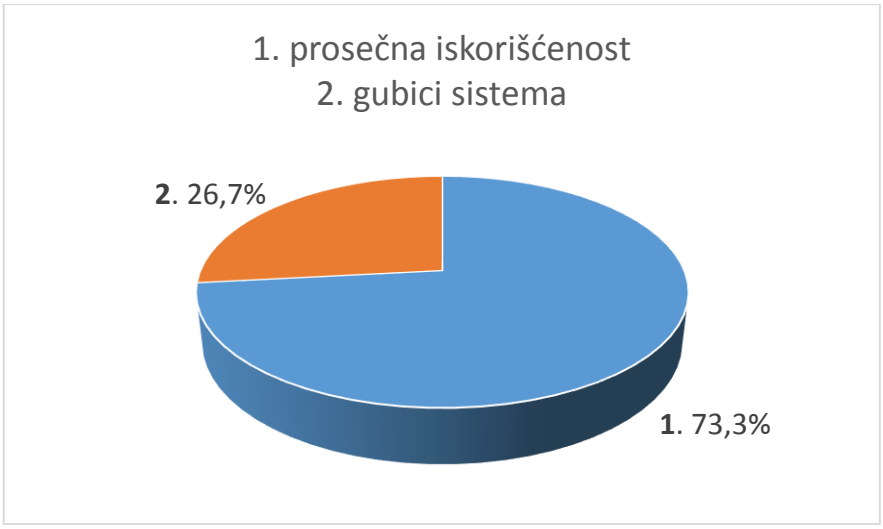
Na osnovu prikupljenih i obrađenih podataka o intezitetu dolaznog toka korisnika, raspodelama vremena opsluge, zastupljenosti pojedinih usluga i uočenih specifičnosti u opsluživanju klijenata na šalterima novčanog poslovanja i šalterima za prijem poštanskih pošiljaka formirani su simulacioni modeli i izgrađeni simulacioni programi korišćenjem objektno orijentisanog jezika C++.

Kao vremenska jedinica u modelima je korišćena 1s (sekunda). Da bi se dobila veća tačnost izlaznih parametara sistema, simulacija je vršena sa osam generatora slučajnih brojeva za šalter novčanog poslovanja i sa sedam generatora slučajnih brojeva za šalter poštanskih pošiljaka. Prilikom formiranja programa vođeno je računa o obezbeđivanju statističke nezavisnosti. Izvori slučajnosti su statistički nezavisni što znači da promene koje se dešavaju u jednom delu ne utiču na ponašanje drugih delova modela. Generatori slučajnih brojeva se koriste pri generisanju dolaznog toka klijenata, vremena opsluživanja, izbora usluge i broja usluga koji svaki korisnik zahteva. Ovi procesi se u realnosti dešavaju apsolutno nezavisno jedan od drugog, pa ta karakteristika treba biti preneta i u model. Korišćenjem različitih generatora pseudoslučajnih brojeva uklanja se zavisnost i verno oponaša realni sistem. U slučaju programskom jezika C++ generatore slučajnih brojeva pozivamo kao objekte klase „raspodela“ sa različitim semenima generisanja. U koloni kombinacija tokova generatora slučajnih brojeva tabela 6.5.1 i 6.5.2 navodimo semena generatora iz koda programskog jezika C++, za svaku realizaciju programa, respektivno, kako smo definisali objekte klase “raspodela”u kodu.

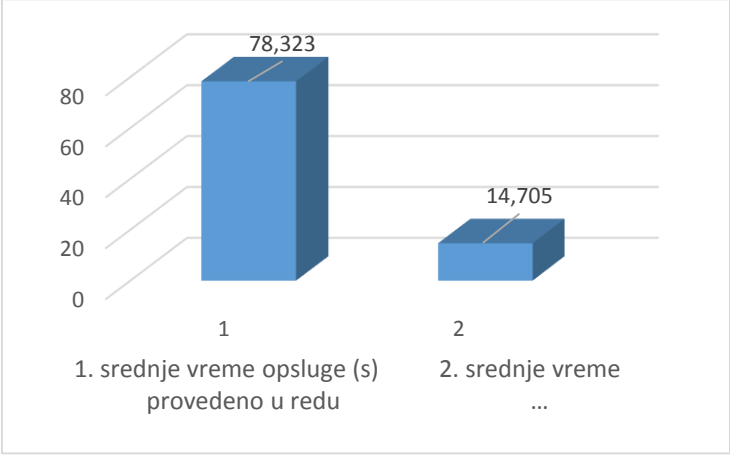
Prosečno trajanje opsluge na šalteru novčanog poslovanja je 65,83s i čekali su 116,31s, dok su se na drugom šalteru opsluživali 78,32s i čekali u redu 14,71. Na osnovu ovih podataka iz tabela 6.5.1 i 6.5.2 možemo da zaključimo da nije zadovoljen apsolutni kriterijum kvaliteta opsluživanja na šalteru novčanog poslovanja, dok je na drugom šalteru ovaj uslov zadovoljen. Stepen zauzetosti šaltera novčanog poslovanja, odnosno njegova iskorišćenost jeste 73,3 %, a gubitak sistema 26,7%, dok je na šalteru za pismonosne i paketske usluge 21,6% iskorišćenost, a 78,4% gubitak sistema. Na slikama 6.5.1 i 6.5.3 su prikazani odnosi odnosi srednjih vremena opsluge i čekanja dok je na slici 6.5.2 predstavljen je odnos iskorišćenosti i gubitka sistema na šalteru novčanog poslovanja.



Slika 6.5.1. Odnos srednjeg vremena opsluge i srednjeg vremena čekanja u redu na šalteru novčanog poslovanja



Slika 6.5.2. Odnos prosečnog efektivnog vremena rada i gubitka sistema na šalteru novčanog poslovanja



Slika 6.5.3.Odnos srednjeg vremena opsluge i srednjeg vremena provedenog u redu čekanja na šalteru poštanskih pošiljaka

Tabela 6.5.1. Rezultati simulacije postojećeg stanja šalterske službe na šalteru novčanog poslovanja

Kombinacije tokova generatora slučajnih brojeva	Iskorišćenost kanala opsluživanja	Srednja dužina reda čekanja	Procenat klijenata koji ne čekaju u redu	Srednje vreme opsluge (s)	Srednje vreme provedeno u redu (s)
rn1(1)	72,4	1,610	28,2	72,000	155,270
rn2(3)	67,0	0,668	41,0	63,180	58,500
rn3(5)	80,4	0,877	28,3	63,368	68,650
rn4(7)	77,9	1,479	20,9	72,275	111,570
rn5(11)	90.7	2,880	11,8	69,056	207,000
rn6(13)	76,0	1,530	19,6	59,934	119,000
rn7(17)	66,1	1,021	33,3	61,003	94,207
srednje vrednosti	73,3	1,438	26,2	65,830	116,314

Tabela 6.5.2. Rezultati simulacije postojećeg stanja šalterske službe na šalteru za pošiljke

Kombinacije tokova generatora slučajnih brojeva	Iskorišćenost kanala opsluživanja	Srednja dužina reda čekanja	Procenat klijenata koji ne čekaju u redu	Srednje vreme opsluge (s)	Srednje vreme provedeno u redu (s)
rn1(1)	19,1	0,019	90,9	62,743	6,416
rn2(3)	23,7	0,069	80,0	85,349	25,120
rn3(5)	15,7	0,057	77,8	63,019	23,181
rn4(7)	20,8	0,025	85,7	105,643	13,050
rn5(11)	31,7	0,125	66,7	76,229	30,000
rn6(13)	13,2	0,000	100,0	67,453	0,000
rn7(17)	26,8	0,016	90,9	87,823	5,170
srednje vrednosti	21,6	0,044	84,6	78,323	14,705

Na osnovu dobijenih podataka može se zaključiti da opsluživanje korisnika nije kvalitetno na šalteru novčanog poslovanja po apsolutnom kriterijumu ali i po kriterijumu da je srednja dužina reda čekanja kraće od broja šaltera. Za šalter za pismonosne i paketske usluge možemo da kažemo da ima kvalitetnu opslugu, na osnovu oba, prethodno navedena kriterijuma. Na osnovu iskorišćenosti oba šaltera kao modifikaciju modela možemo da realizujemo programa tako što će u zavisnosti od broja korisnika u redu čekanja za novčano poslovanje, vršiti preusmeravanje određenog broja korisnika na šalter za pakete gde će oni biti opsluženi.

6.6.Predlog poboljšanja i modifikacija modela

Modifikaciju u modelu predstavlja preusmeravanje korisnika sa šaltera novčanog poslovanja na šalter za prijem pošiljaka, kada je red čekanja na prvom šalteru veći od tri korisnika.

Tabela 6.6.1. Rezultati simulacije modifikovanog stanja šalterske službe na šalteru novčanog poslovanja

Kombinacije tokova generatora slučajnih brojeva	Iskorišćenost kanala opsluživanja	Srednja dužina reda čekanja	Procenat klijenata koji ne čekaju u redu	Srednje vreme opsluge (s)	Srednje vreme provedeno u redu (s)
rn1(1)	60,0	0,473	35,9	66,000	44,000
rn2(3)	65,2	0,543	41,0	62,760	46,670
rn3(5)	75,4	0,945	28,3	65,505	74,027
rn4(7)	76,2	1,072	20,9	75,590	90,012
rn5(11)	83,9	0,973	23,5	70,340	66,918
rn6(13)	64,7	0,840	21,7	61,348	65,806
rn7(17)	60,1	0,441	38,5	60,104	40,742
srednje vrednosti	69,4	0,755	30,0	65,950	61,168

Tabela 6.6.2. Rezultati simulacije modifikovanog stanja šalterske službe na šalteru za pošiljke

Kombinacije tokova generatora slučajnih brojeva	Iskorišćenost kanala opsluživanja	Srednja dužina reda čekanja	Procenat klijenata koji ne čekaju u redu	Srednje vreme opsluge (s)	Srednje vreme provedeno u redu (s)
rn1(1)	30,2	0,308	66,7	72,520	73,960
rn2(3)	25,8	0,087	72,7	84,680	28,610
rn3(5)	22,9	0,066	61,5	63,688	18,381
rn4(7)	27,1	0,175	58,3	86,755	52,570
rn5(11)	43,1	0,279	57,1	73,980	47,570
rn6(13)	27,5	0,040	73,3	66,207	9,659
rn7(17)	41,8	0,251	64,3	107,590	64,650
srednje vrednosti	31,2	0,172	64,800	73,346	42,200

Iz tabela 6.6.1 i 6.6.2 možemo da vidimo, da ukoliko izvršimo preusmerenje korisnika koji se nalaze u redu čekanja na šalteru novčanog poslovanja na red čekanja i šalter za pošiljke, ispunjava se uslov za kvalitetnu opslugu na osnovu apsolutnog kriterijuma.

Kod šaltera za novčane usluge srednje vreme opsluge (65,95s) veće od srednjeg vremena provedenog u redu čekanja na istom šalteru (61,168s). Takođe je i srednja dužina reda manja od samog broja šaltera (u našem slučaju 1 šalter), i ona iznosi 0,755. Što se ti šaltera za pošiljke zaključak je sličan. Srednje vreme opsluge (73,346s) je veće od srednjeg vremena provedenog u redu čekanja (42,2s). Srednja dužina reda iznosi 0,172, pa je i na osnovu tog kriterijuma opsluga na šalteru za pošiljke kvalitetna. U prilogu završnog rada nalazi se modifikovan model, koji preusmerava sa šaltera novčanih usluga na šalter za pošiljke kada je red čekanja veći od tri korisnika. Na slikama 6.6.1 i 6.6.2 predstavljamo iskorišćenosti oba šaltera iskazani u procentima.

Slika 6.6.1. Odnos prosečnog efektivnog vremena rada i prosečnog vremena praznog hoda na šalteru uplate – isplate

Slika 6.6.2. Odnos prosečnog efektivnog vremena rada i prosečnog vremena praznog hoda na šalteru za pošiljke

7. ZAKLJUČAK

Cilj završnog rada jeste analiziranje funkcionisanja šalterske službe pošte 11158 Beograd i utvrđivanje nivoa kvaliteta pružanja usluga korisnicima. Analiza sistema je vršena uz pomoć objektno orijentisanog programskog jezika C++. Eksperimetisanjem nad formiranim simulacionim modelima dobijeni su parametri sistema koji pokazuju stepen njegove iskorišćenosti, verovatnoće postojanja redova, dužine redova čekanja, vremena čekanja korisnika u redu, vremena njihovog opsluživanja. Analiza kvaliteta opsluživanja korisnika koji pristupaju šalteru novčanog poslovanja i šalteru poštanskih pošiljaka vršena je na osnovu apsolutnog kriterijuma i na osnovu dužine redova čekanja u odnosu na kapacitete sistema.

Na osnovu rezultata simulacije za postojeće stanje organizacije šalterske službe utvrđeno je da kvalitet opsluživanja korisnika nije zadovoljavajući i to na osnovu dva kriterijuma. Iskorišćenost sistema za postojeće stanje šaltera respektivno, je 73,3% i 21,6%. Prosečno vreme čekanja i opsluge na šalter novčanog poslovanja jesu 116,3s i 65,8s i vidimo da apsolutni kriterijum nije zadovoljen. Pored toga srednja dužina reda iznosi 1,438. Za šalter poštanskih pošiljaka vremena su 14,7s i 78,2s gde primećujemo da je apsolutni kriterijum zadovoljen.

S obzirom da pošta dobija najveći prihod od novčanog poslovanja, time zaključujemo da je usluga novčanog poslovanja traženija u odnosu na pimonosno - paketske usluge i samim tim na šalteru se javlja veći broj korisnika u odnosu na šalter pismonosno – paketskih usluga. Iz tog razloga kao modifikaciju modela navodimo preusmeravanje korisnika sa šaltera novčanog poslovanja na šalter poštanskih pošiljaka, u slučaju da u redu postoji više od 3 korisnika.

Rezultati simulacije modifikovanog modela pokazuju da je kvalitet opsluge korisnika na zadovoljavajućem nivou. Apsolutni kriterijum je zadovoljen jer je šalteru novčanog poslovanja prosečno vreme čekanja 61,2 s, dok je srednje vreme opsluge 65,9s. Dužina reda iznosi 0,755 i iskorišćenost šaltera iznosi 69,4%. Za šalter za pisma i pakete srednje vreme čekanja iznosi 42,2s, srednje vreme opsluge 73,5s i na osnovu tih rezultata, apsolutni kriterijum je zadovoljen. Samim tim povećali smo i procenat iskorišćenosti koji iznosi 31,2%.

Na osnovu predložene modifikacije modela predlog poboljšanja kvaliteta opsluge korisnika jeste da kada se napravi red čekanja korisnika na šalteru novčanog poslovanja koji je veći od tri korisnika u redu, preusmeravaju korisnici na šalter poštanskih usluga na način da se prvi korisnik iz reda preusmerava, pa po njegovom opsluživanju i da se naredni prvi korisnik koji čeka u redu na šalteru novčanog poslovanja preusmeri kada je red čekanja veći od tri korisnika.

LITERATURA

[1] S. Vukadinović, "Elementi teorije masovnog opsluživanja", Saobraćajni fakultet, Beograd, 1983.

[2] M. Stanojević, "Simulaciona analiza funkcionisanja šalterske službe u PTT-u", Četrnaesti naučno stručni simpozijum o novim tehnologijama u PTT-u, Zbornik radova, Saobraćajni fakultet, Beograd, 1996.

[3] B: Radenković, M: Stanojević, A. Marković, "Računarska simulacija", Saobraćajni fakultet, Fakultet organizacionih nauka, Beograd, 2009.

[4] M. Stanojević, "Mesto i uloga modeliranja u oblasti PTT-a", Deseti naučno stručni simpozijum za inovacju znanja u oblasti novih tehnologija u PTT-u, Zbornik radova, Saobraćajni fakultet, Beograd, 1992.

[5] M. Đogatović, "Simulacione strategije", Seminarski rad, Saobraćajni fakultet, Univezitet u Beogradu, 2010.

[6] K. Iker, "Objektno orijentisana simulacija i animacija na primeru sistema za razvrstavanje paketa", Završni rad; Saobraćajni fakultet, Beograd, 2014.

[7] S. Malkov, "C++ kroz primere", Odlomci iz knjige u primpremi, Matematički fakultet, Univeziteta u Beogradu, 2005.

[8] B. Struostrup, "The C++ programming Language", 3rd edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1997.

[9] S. Lippman, J. Lajoie, B.Moo, "C++ Primer", 5th edition, Addison-Wesley Publishing Company, Westford, Massachusetts, 2013.

[10] J. Liberty, "Teach Yourself C++ in 21 Days", 2nd edition, SAMS Publishing, Indianapolis, 1997.

[11] S. Ilić, "Simulaciona analiza funkcionisanja šalterske službe u pošti 11010 Beograd", Završni rad, Saobraćajni fakultet, Beograd, 2012.

[12] J. Tomašević, "Osnovi računarskih sistema – STL", Matematički fakultet, Beograd, 2005.

[13] D. Marković, B. Grgurović, " Poštanski saobraćaj", Saobraćajni fakultet, Beograd, 2006.

PRILOG

Datoteka main.cpp

```
// Datoteka simul.h
#include "simul.h"
#include <iostream>
#include <string>
#include <iomanip>
// Makro naredbe
#define BROJ_USLUGA 5
#define VRSTA_USLUGE 1
#define VREME simu.vrati_vreme_simulacije()
#define VRTR(vreme) simu.vrati_vreme_simulacije()+(vreme)
#define TERMINATE(vrednost) simu.unisti_entitet(e, vrednost)

// Ukljucujemo prostore imena simul i std
using namespace simul;
using namespace std;

// Objekat simulacije
simulacija simu;
// Raspodele
raspodela rn1(1), rn2(3), rn3(7), rn4(9), rn5(11), rn6(15),
rn7(17),rn8(19);
// Redovi cekanja
red_cekanja red_uni(&simu), red_pak(&simu);
// Salteri
resurs uni(&simu, 1),pak(&simu, 1);
// Funkcije
funkcija func;
//verovatnoca kojom korisnik dolazi u postu u odredjenom vremenskom
intervalu
double ver_dol_uni[12] = { 0.000, 0.208, 0.399, 0.486, 0.601, 0.650,
0.715,
0.775, 0.824, 0.890, 0.945, 1.000 };
//vremenski interval u kom dolazi korisnik
double vre_dol_uni[12] = { 0.00, 20.00, 40.00, 60.00, 80.00, 100.00,
120.00,
140.00, 180.00, 200.00, 220.00, 380.00};
//funkcija kojom se vraca realno vreme u koje korisnik dolazi
double vrati_vre_dol_uni(){
      return func.kontinualna(rn7(), ver_dol_uni, vre_dol_uni, 12);
}
//verovatnoca izbora usluge
double izb_usl_uni[BROJ_USLUGA] = { 0.277, 0.447, 0.661, 0.843, 1.00 };
//vrsta usluge
double vrs_usl_uni[BROJ_USLUGA] = { 1.00, 2.00, 3.00, 4.00, 5.00 };
//funkcija kojom vracamo vrednost usluge koju smo izabrali
double izaberi_usl_uni(){
      //vracamo disretnu vrednost, jer se radi o vrsti usluge
      return func.diskretna(rn8(), izb_usl_uni, vrs_usl_uni,
BROJ_USLUGA);
}
//verovatnoca kojom se bira za koje vreme se korisnik opsluzuje ukoliko
odabere prvu uslugu
double ops_1[5]={0.000,0.414, 0.771, 0.942, 1.000};
// vreme za koje se opsluzi korisnik
double vremena_ops1[5]={ 30.00, 60.00, 90.00, 120.00, 150.00};
```

```
//verovatnoca kojom se bira za koje vreme se korisnik opsluzuje ukoliko
odabere drugu uslugu
double ops_2[4]={0.000, 0.605, 0.907, 1.000};
// vreme za koje se opsluzi korisnik
double vremena_ops2[4]={ 20.00, 40.00, 60.00, 80.00};
//verovatnoca kojom se bira za koje vreme se korisnik opsluzuje ukoliko
odabere trecu uslugu
double ops_3[5]={0.000, 0.518, 0.777, 0.944, 1.000};
// vreme za koje se opsluzi korisnik
double vremena_ops3[5]={ 20.00, 40.00, 60.00, 80.00, 100.00};
//verovatnoca kojom se bira za koje vreme se korisnik opsluzuje ukoliko
odabere cetvrtu uslugu
double ops_4[5]={0.000, 0.435, 0.783, 0.956, 1.000};
// vreme za koje se opsluzi korisnik
double vremena_ops4[5]={ 30.00, 60.00, 90.00, 120.00, 150.00};
//verovatnoca kojom se bira za koje vreme se korisnik opsluzuje ukoliko
odabere petu uslugu
double ops_5[4]={0.000, 0.469, 0.836, 1.000};
// vreme za koje se opsluzi korisnik
double vremena_ops5[4]={ 20.00, 40.00, 60.00, 80.00};
//ukoliko korisnik zeli vise transakcija odjednom
double koliko_usluga[2]={1.,2.};
//verovatnoca izbora jedne ili dve transakcije
double ver_koliko_usluga[2] = {0.675,1.0};
//funkcija kojom vracamo koliko transakcija zelimo
double broj_usluga(){
      return func.diskretna(rn1(), ver_koliko_usluga,koliko_usluga,2);
}
//funkcija kojom vracamo realno vreme za koje se korisnik opsluzi
double vrati_vre_ops_uni(const double &vrsta_usluge){
      //u zavisnosti od usluge koju je odabrao
      switch (int(vrsta_usluge)){
      case 1:
            return
(func.kontinualna(rn2(),ops_1,vremena_ops1,5)*broj_usluga());
      case 2:
            return
(func.kontinualna(rn3(),ops_2,vremena_ops2,4)*broj_usluga());
      case 3:
            return
(func.kontinualna(rn4(),ops_3,vremena_ops3,5)*broj_usluga());
      case 4:
            return
(func.kontinualna(rn5(),ops_4,vremena_ops4,5)*broj_usluga());
      case 5:
            return
(func.kontinualna(rn6(),ops_5,vremena_ops5,4)*broj_usluga());
      }
      return 0;
}
//verovatnoca kojom korisnici dolaze na paketski salter
double ver_dol_pak[5] = { 0.000, 0.417, 0.722, 0.861, 1.000 };
//vremenski intervali u kojim korisnik dolazi
double vre_dol_pak[5] = {  0.00, 200.00, 400.00, 800.00, 1000.00};
//funkcija kojom se vraca realno vreme dolazaka korisnika
double vrati_vre_dol_pak(){
      return func.kontinualna(rn7(), ver_dol_pak, vre_dol_pak, 5);
```

```
}
//verovatnoca izbora neke od pismonosno-paketskih usluga
double izb_usl_pak[BROJ_USLUGA] = { 0.440, 0.678, 0.845, 0.928, 1.00 };
//konkretna vrsta usluge
double vrsta_usl_pak[BROJ_USLUGA] = { 1, 2, 3, 4, 5 };
//funkcija koja vracamo celobrojnu vrednost za izabranu uslugu
int izaberi_usl_pak(){
      return int(func.diskretna(rn1(), izb_usl_pak, vrsta_usl_pak,
BROJ_USLUGA));
}
//verovatnoca vremenskog intervala opsluge prve usluge
double ops_11[4]={0.000,0.156, 0.565, 1.000};
//realno vreme opsluge
double vremena_ops11[4]={ 20.00, 30.00, 60.00, 90.00};
//verovatnoca vremenskog intervala opsluge druge usluge
double ops_22[3]={ 0.500, 0.833, 1.000};
//realno vreme opsluge
double vremena_ops22[3]={ 120.00, 180.00, 240.00};
//verovatnoca vremenskog intervala opsluge trece usluge
double ops_33[3]={0.200, 0.900, 1.000};
//realno vreme opsluge
double vremena_ops33[3]={ 30.00, 60.00, 90.00};
//verovatnoca vremenskog intervala opsluge cetvrte usluge
double ops_44[2]={ 0.571, 1.000};
//realno vreme opsluge
double vremena_ops44[2]={60.00, 120.00};
//verovatnoca vremenskog intervala opsluge pete usluge
double ops_55[2]={ 0.167, 1.000};
//realno vreme opsluge
double vremena_ops55[2]={ 90.00, 120.00};
//funkcija kojom se vraca realno vreme opsluge
double vrati_vre_ops_pak(const double &vrsta_usluge){
      switch (int(vrsta_usluge)){
      case 1:
            return func.kontinualna(rn2(),ops_11,vremena_ops11,4);
      case 2:
            return func.diskretna(rn3(),ops_22,vremena_ops22,3);
      case 3:
            return func.diskretna(rn4(),ops_33,vremena_ops33,3);
      case 4:
            return func.diskretna(rn5(),ops_44,vremena_ops44,2);
      case 5:
            return func.diskretna(rn6(),ops_55,vremena_ops55,2);
      }
      return 0;
}
//konstante koje govore o kom se dogadjaju radi
const int DOLAZAK_NA_UNIVE = 1;
const int ODLAZAK_SA_UNIVE = 2;
const int DOLAZAK_NA_PAKET = 3;
const int ODLAZAK_SA_PAKET = 4;
const int KRAJ_SIMULACIJE = 5;
//funkcije koje se pozivaju kada odgovaraju broju dogadjaja
void dolazak_na_unive(entitet *e);
void odlazak_sa_unive(entitet *e);
void dolazak_na_paket(entitet *e);
void odlazak_sa_paket(entitet *e);
```

```
void kraj_simulacije(entitet *e);
//definicija metode izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
	//u zavisnosti od dogadjaja
	switch (dog) {
	case DOLAZAK_NA_UNIVE:
		dolazak_na_unive(e);
		break;
	case ODLAZAK_SA_UNIVE:
		odlazak_sa_unive(e);
		break;
	case DOLAZAK_NA_PAKET:
		dolazak_na_paket(e);
		break;
	case ODLAZAK_SA_PAKET:
		odlazak_sa_paket(e);
		break;
	case KRAJ_SIMULACIJE:
		kraj_simulacije(e);
		break;
	}
}
//ukoliko se javi dog==1
void dolazak_na_unive(entitet *e)
{
	entitet *fe, *ne;
	// Postavi klijenta u red cekanja
	red_uni.ured(e);
	//MODIFIKACIJA MODELA
//ukoliko se pojavi vise od tri korisnika nu redu uplate-isplete
	if(red_uni.velicina()>3){
		//vadimo prvog iz univerzalnog
			fe=red_uni.izred();
//smestamo tok korisnika u paketski red
			red_pak.ured(fe);
//pitamo da li je salter pak slobodan
			if(pak.raspoloziv()){
//ako jeste, onda vadimo prvog iz pak reda
				fe = red_pak.izred();
//zauzimamo pak salter
				pak.zauzmi(fe);
simu.rasporedi(fe, ODLAZAK_SA_PAKET,
VRTR(vrati_vre_ops_uni(fe>vrati_parametar(VRSTA_USLUGE))));
			}
		}
	// Ukoliko je salter raspoloziv
	if (uni.raspoloziv()) {
		// Izvadi prvog klijenta iz reda
		fe = red_uni.izred();
		// Zauzmi jedno mesto u salteru
		uni.zauzmi(fe);
		// Rasporedi klijenta za kraj opsluge
		simu.rasporedi(fe, ODLAZAK_SA_UNIVE,
VRTR(vrati_vre_ops_uni(fe->vrati_parametar(VRSTA_USLUGE))));
	}
```

```
	// Stvaramo narednog klijenta
	ne = simu.napravi_entitet();
	// Dodeljujemo vrstu usluge
	ne->postavi_parametar(VRSTA_USLUGE, izaberi_usl_uni());
	// Postavljamo vreme narednog dolaska
	simu.rasporedi(ne, DOLAZAK_NA_UNIVE, VRTR(vrati_vre_dol_uni()));
}
// Dogadjaj odlazak klijenta
void odlazak_sa_unive(entitet *e) {
	entitet *fe;
	//verovatnoca kojom se korisnici sa univerzalnog saltera
preusmeravaju na paketski
	// Vrati salter;
	uni.oslobodi(e);
	// Klijent odlazi iz poste - unistavamo tekuceg klijenta
	simu.unisti_entitet(e, 0);
	// Ukoliko red nije prazan
	if (red_uni.velicina()>0) {
		// Izvadi prvog klijenta iz reda
		fe = red_uni.izred();
		// Zauzmi jedno mesto u salteru
		uni.zauzmi(fe);
		// Rasporedi klijenta za kraj opsluge
simu.rasporedi(fe, ODLAZAK_SA_UNIVE,
VRTR(vrati_vre_ops_uni(fe>vrati_parametar(VRSTA_USLUGE))));
		}
}
void dolazak_na_paket(entitet *e)
{
	entitet *fe, *ne;
	// Postavi klijenta u red cekanja
	red_pak.ured(e);
	// Ukoliko je salter raspoloziv
	if (pak.raspoloziv()) {
		// Izvadi prvog klijenta iz reda
		fe = red_pak.izred();
		// Zauzmi jedno mesto u salteru
		pak.zauzmi(fe);
		// Rasporedi klijenta za kraj opsluge
		simu.rasporedi(fe, ODLAZAK_SA_PAKET,
VRTR(vrati_vre_ops_pak(fe->vrati_parametar(VRSTA_USLUGE))));
	}
	// Stvaramo narednog klijenta
	ne = simu.napravi_entitet();
	// Dodeljujemo vrstu usluge
	ne->postavi_parametar(VRSTA_USLUGE, izaberi_usl_pak());
	// Postavljamo vreme narednog dolaska
	simu.rasporedi(ne, DOLAZAK_NA_PAKET, VRTR(vrati_vre_dol_pak()));
}
// Dogadjaj odlazak klijenta
void odlazak_sa_paket(entitet *e) {
	entitet *fe;
	// Vrati salter;
	pak.oslobodi(e);
```

```
	// Klijent odlazi iz poste - unistavamo tekuceg klijenta
	simu.unisti_entitet(e, 0);
	// Ukoliko red nije prazan
	if (red_pak.velicina()>0) {
		// Izvadi prvog klijenta iz reda
		fe = red_pak.izred();
		// Zauzmi jedno mesto u salteru
		pak.zauzmi(fe);
		// Rasporedi klijenta za kraj opsluge
simu.rasporedi(fe, ODLAZAK_SA_PAKET,
VRTR(vrati_vre_ops_pak(fe>vrati_parametar(VRSTA_USLUGE))));
	}
}
// Dogadjaj kraj simulacije
void kraj_simulacije(entitet* e) {
	//unistavamo entitete, uklanjanje
	TERMINATE(1);
}
void print_statistike_reda_cekanja(const red_cekanja& red, const char*
opis) {
	statistike stat = red.vrati_statistike();
	// opis parametara redova cekanja
	cout << "STATISTIKE REDA CEKANJA - " << opis << endl;
	cout << "Broj entiteta koji je usao u red cekanja: " <<
stat.vrati_broj() << endl;
	cout << "Preostali broj entiteta u redu cekanja: " <<
stat.vrati_tekuci_broj() << endl;
	cout << "Maksimalni broj entiteta u redu: " <<
stat.vrati_maksimalni_broj() << endl;
	cout << "Srednje vreme cekanja u redu: " <<
stat.vrati_srednje_vreme() << endl;
	cout << "Srednji broj entiteta u redu: " <<
stat.vrati_srednji_broj(simu.vrati_vreme_simulacije()) << endl;
	cout << "Broj entiteta koji je prosao kroz red bez zadrzavanja: "
<< stat.vrati_broj_bez_cekanja() << endl;
	cout << "Procenat ulaza bez zadrzavanja: " <<
stat.vrati_ucesce_bez_cekanja() << endl;
	cout << "Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu
zadrzali): " << stat.vrati_srednje_vreme_bez_cekanja() << endl;
	cout << endl;
}

void print_statistike_resursa(const resurs& salt, const char* opis) {
statistike stat = salt.vrati_statistike();
// opis parametara koje prikazujemo za svaki salter
cout << "STATISTIKE RESURSA - " << opis << endl;
cout << "Broj entiteta koji je usao u resurs: " << stat.vrati_broj() <<
endl;
cout << "Preostali broj entiteta u resursu: " << stat.vrati_tekuci_broj()
<< endl;
cout << "Maksimalni broj zauzetih kanala opsluge: " <<
stat.vrati_maksimalni_broj() << endl;
cout << "Srednje vreme opsluge: " << stat.vrati_srednje_vreme() << endl;
cout << "Srednji broj zauzetih kanala: " <<
stat.vrati_srednji_broj(simu.vrati_vreme_simulacije()) << endl;
cout << "Iskoriscenost: " <<
stat.vrati_iskoriscenost(simu.vrati_vreme_simulacije(),
```

```
salt.vrati_broj_mesta()) << endl;
cout << endl;
}
//main program
int main()
{
//rasporedjujemo klijenta na dolazak na salter
simu.rasporedi(simu.napravi_entitet(), DOLAZAK_NA_UNIVE,
vrati_vre_dol_uni());
//rasporedujemo klijenta na dolazak na salter
simu.rasporedi(simu.napravi_entitet(), DOLAZAK_NA_PAKET,
vrati_vre_dol_pak());
//rasporedjujemo korisnika za kraj simulacije
simu.rasporedi(simu.napravi_entitet(), KRAJ_SIMULACIJE, 3600);
simu.izvrsi(1);
red_uni.finisiraj();
red_pak.finisiraj();
uni.finisiraj();
pak.finisiraj();
// Stampanje statistika za redove cekanja i salter uni
print_statistike_reda_cekanja(red_uni, "RED CEKANJA NA UISAL");
print_statistike_resursa(uni, "UPLATE_ISPLATE SALTER");
//stampanje statistika za redove cekanja i saltere pak
print_statistike_reda_cekanja(red_pak, "RED CEKANJA NA PAKSAL");
print_statistike_resursa(pak, "PAKETSKI SALTER");
//vreme simulacije
cout << "Vreme simulacije: " << VREME << endl << endl;

system("pause");
return 0;
}
```