



Funded by the
Erasmus+ Programme
of the European Union



This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Arhitektura softvera

Lekcija 04

PRIRUČNIK ZA STUDENTE

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Lekcija 04

ARHITEKTURA SOFTVERA

- ✓ Arhitektura softvera
- ✓ Poglavlje 1: Arhitektura sistema
- ✓ Poglavlje 2: Odluke u vezi arhitekture
- ✓ Poglavlje 3: Projektovanje arhitekture sistema
- ✓ Poglavlje 4: Arhitektonski šabloni
- ✓ Poglavlje 5: Arhitekture aplikacija
- ✓ Poglavlje 6: Projektna dokumentacija
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

✓ Uvod

UVOD

Cilj nastavne jedinice

Cilj ove lekcije jeste da vas upozna sa konceptom arhitekture softvera i projektnim rešenjem arhitekture. Ova lekcija vam omogućava da:

- razumete zašto je projektno rešenje arhitekture softvera važno;
- razumete odluke koje treba doneti o arhitekturi sistema za vreme procesa projektovanja arhitekture;
 - se upoznate sa idejom arhitektonskih mustri/šablona (eng., patterns), tj. oprobanog načina organizovanja arhitekture sistema, a koje se mogu upotrebiti u projektima sistema;
- saznote arhitektonske šablone softvera koje se često koriste kod primenjenih sistema različitog tipa, uključujući sisteme za obradu transakcija i sisteme za obradu jezika.

Kao programeri, od vas se ne očekuje da projektujete softver, pa ni da definišete njegovu arhitekturu. Međutim, kako ćete učestvovati u projektnom timu za razvoj softvera, potrebno je da razumete proces njegovog razvoja, aktivnosti tog procesa, i dokumentaciju koju on proizvodi. Zato, ova nastavna jedinica vam daje minimalno potrebna znanja iz arhitekture softverskih sistema, a neophodna je razumevanje pojmove i problematike.

✓ Poglavlje 1

Arhitektura sistema

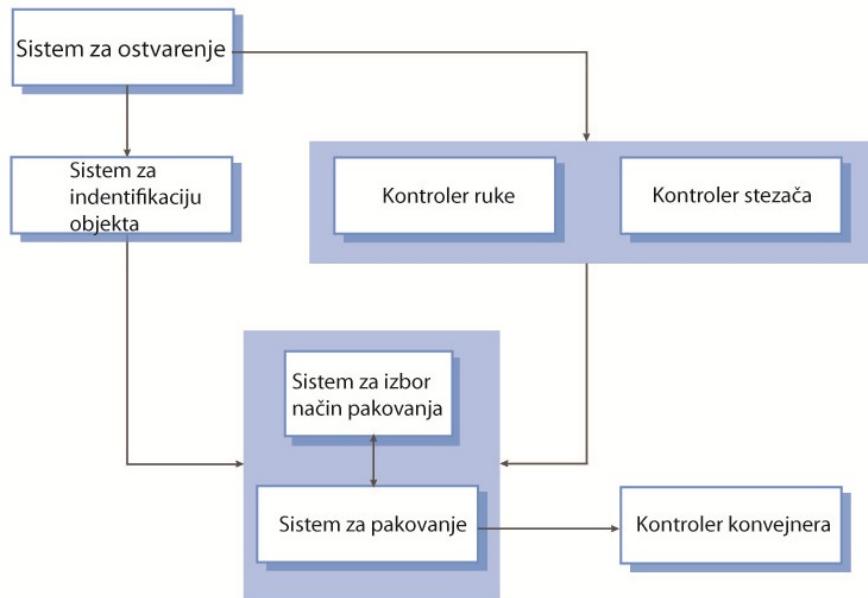
ŠTA JE ARHITEKTURA SOFTVERA?

Arhitektura softvera ukazuje nam kako bi sistem trebalo da bude organizovan i prikazuje ukupnu strukturu sistema.

Arhitektura softvera ukazuje nam kako bi sistem trebalo da bude organizovan i prikazuje ukupnu strukturu sistema. Projektovanje arhitekture je prva faza procesa projektovanja softvera. To je kritična veza između inženjerstva zahteva i projektovanja softvera, jer utvrđuje glavne strukturne komponente u sistemu i veze između njih. Na izlazu iz procesa projektovanja arhitekture se dobija model arhitekture koji opisuje kako je sistem organizovan kao skup komunikacionih komponenti.

I kod agilnih procesa se smatra da je u ranim fazama procesa razvoja neophodno da se definiše ukupna arhitektura sistema. Inkrementalni razvoj arhitekture ne vodi uspehu, a promene u arhitekturi sistema su skupe. Zato je važno na početku projektovanja sistema da se postavi kvalitetna arhitektura softverskog sistema koja se kasnije neće menjati. Komponente je lako menjati, ali arhitekturu – nije.

Na slici 1 prikazana je arhitektura jednog robotizovanog sistema za pakovanje. Sistem pakuje različite objekte. Ima komponentu koja mu obezbeđuje da vidi objekt da bi ga uzeo iz konvejera, utvrdio tip objekta i izabrao prvi način pakovanja. Sistem onda prebacuje objekt iz konvejera za isporuku, u konvejer za pakovanje. On stavlja upakovane objekte na drugi konvejer



Slika 1.1 Arhitektura robotizovanog sistema za pakovanje.

POVOLJNOSTI POSTAVLJANJA DOBRE ARHITEKTURE

Dobra arhitektura omogućava dobru komunikaciju sa drugim sistemima i korisnicima, odgovarajući analizu sistema i višestruku upotrebljivost i u drugim softverima.

U idealnom slučaju, specifikacija sistema ne bi trebalo da sadrži informacije o projektnom rešenju softvera. U praksi, sem kod vrlo malih sistema, to se ne može ostvariti. Najčešće je potrebno da se već na početku izvrši dekompozicija arhitekture budućeg sistema da bi se dobila strukturisana i organizovala specifikacija. Zato, već u fazi utvrđivanja zahteva, može se predložiti apstraktna arhitektura sistema, da bi se povezala svojstva sistema sa najvećim i najbitnijim komponentama sistema. Tako predstavljena specifikacija funkcija i zahteva, se onda diskutuje sa akterima sistema.

Projektovanje arhitekture softvera primenjuje dva nivoa apstrakcije:

1. **Arhitektura u malom** – odnosi se na arhitekturu posebnih programa, tj. na strukturu i komponente samog programa.
2. **Arhitektura u velikom** – odnosi se na arhitekturu složenog organizacijskog sistema koji uključuje druge sisteme, programe i programske komponente. Ovo je u stvari arhitektura distribuiranih sistema, što se na Univerzitetu Metropolitan izučava na posebnom predmetu.

Arhitektura softvera je važna jer utiče na performanse, robusnost, distributivnost i održivost sistema. Dok komponente sistema

ostvaruju funkcionalne zahteve sistema, nefunkcionalni zahtevi najviše zavise od arhitekture sistema, jer ona određuje organizaciju tih komponenti i njihovu međusobnu komunikaciju.

Posedovanje projektnog rešenja arhitekture softvera obezbeđuje sledeću povoljnost:

1. *Komunikacija sa akterima sistema:* Arhitektura daje pregled celine sistema i dobro je sredstvo za komunikaciju sa različitim akterima sistema.
2. *Analiza sistema:* Izrada arhitekture sistema u ranim fazama razvoja zahteva sprovođenje odgovarajućih analiza na nivou sistema. Zato, odluke vezane za projektno rešenje arhitekture imaju dubok efekat na to da li će sistem zadovoljiti kritičke zahteve, kao što su performanse, pouzdanost i održivost.
3. *Višestruka upotrebljivost:* Model arhitekture sistema je kompaktan, upravljiv opis kako je sistem organizovan i kako komponente rade zajedno. Arhitektura sistema je često ista za sisteme sa sličnim zahtevima, te se mogu koristiti više puta, kod drugih sličnih sistema.

KORIŠĆENJE MODELA ARHITEKTURE SISTEMA

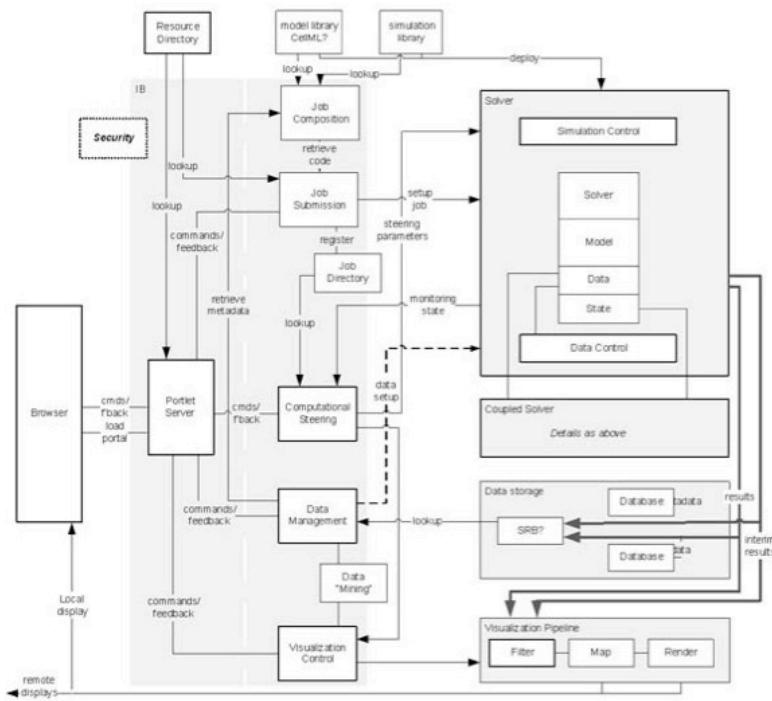
Model arhitekture služi za olakšavanje diskusije o arhitekturi sistema ili za dokumentovanje arhitekture sistema radi razumevanja i daljeg razvoja sistema.

Arhitektura sistema se često modeluje u vidu jednostavnih blok dijagrama, kao na slici 2 , gde svaki pravougaonik predstavlja jednu komponentu. Pravougaonici unutar pravougaonika ukazuju da komponenta ima pod-komponente. Strelice pokazuju tok podataka i upravljačkih signala. Ovako urađeni blok dijagram predstavljaju strukturu sistema koju ljudi iz različitih disciplina, a koji su na neki način deo procesa razvoja sistema, razumeju.

Postoji dva načina korišćenja modela arhitekture nekog programa:

1. *Način za olakšavanje diskusije o projektnom rešenju arhitekture:* Koristi se u razgovorima sa akterima sistema i pri planiranju projekta, jer ne sadrži sve detalje. Model sadrži ključne komponente koje treba razviti, što je dovoljno za planiranje njihovog razvoja.
2. *Način dokumentovanja arhitekture koju treba projektovati:* Cilj ovog načina korišćenja modela je da se proizvede kompletan model sistema kako bi se pokazale različite komponente sistema, njihovi interfejsi, i njihove veze. Detaljan opis arhitekture je potreban radi razumevanja i daljeg razvoja sistema.

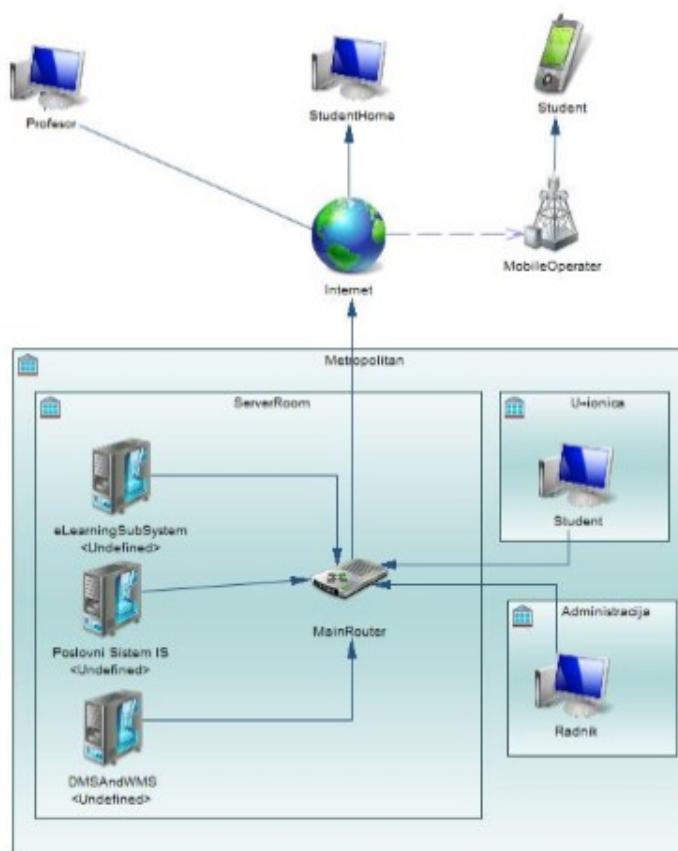
Neki osporavaju korisnost i isplativost primene drugog načina modelovanja arhitekture.



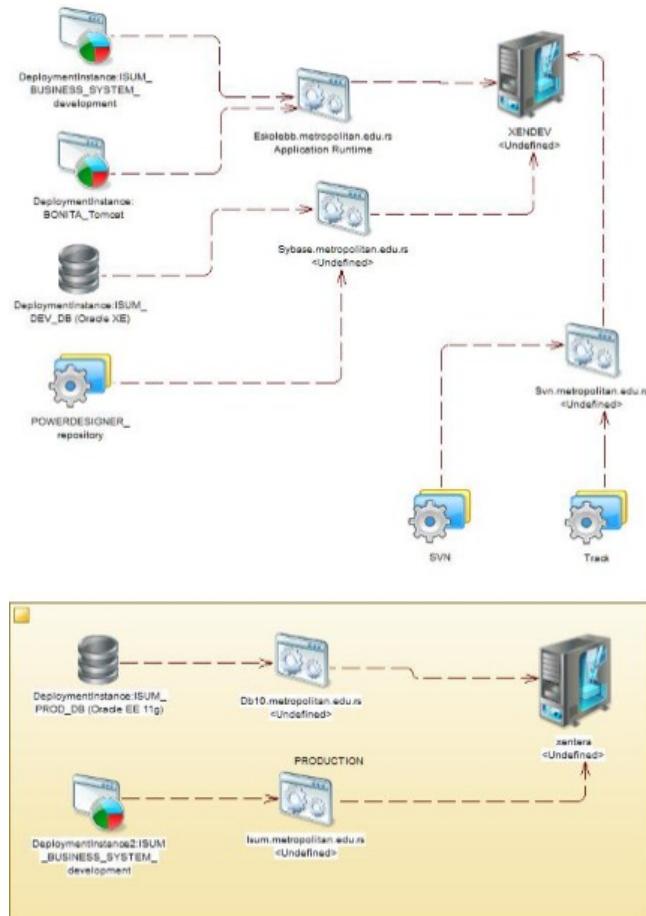
Slika 1.2 Primer jednog projektnog rešenja arhitekture

PRIMER 1

Dijagram arhitekture trebalo bi da se prikaže na jednoj strani



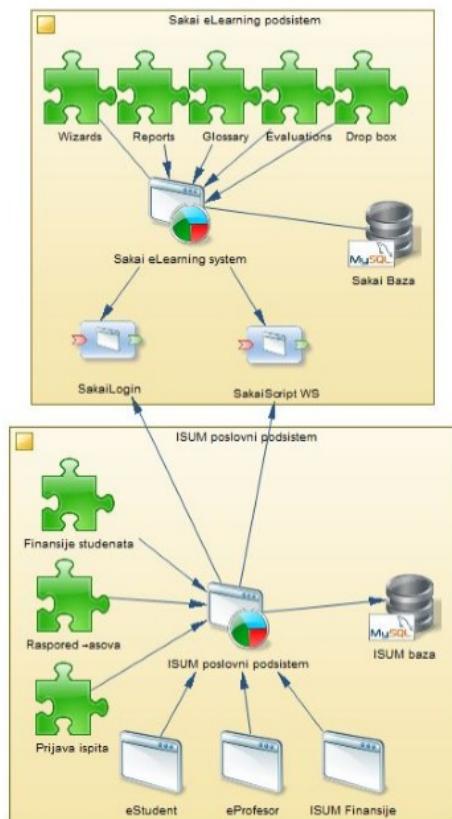
Slika 1.3 Primer jedne arhitekture



Slika 1.4 Primer arhitekture

PRIMER 2

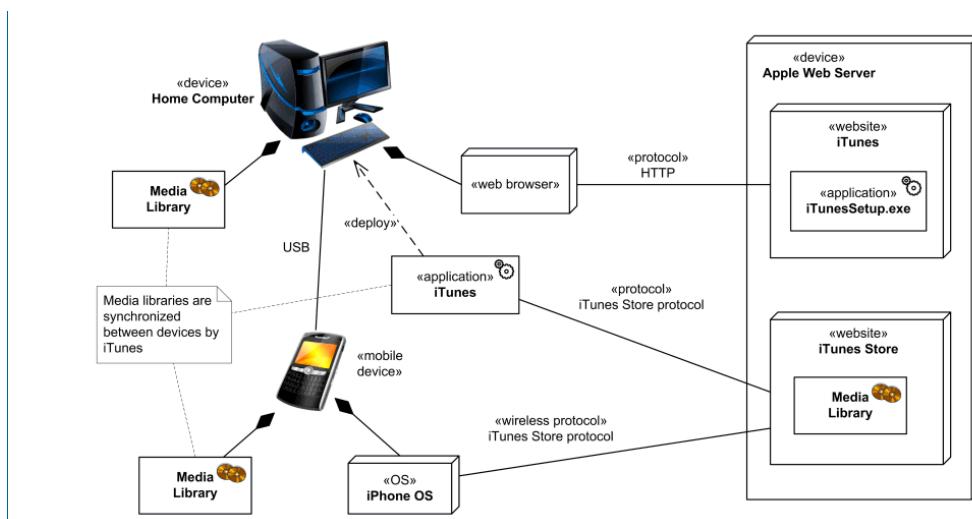
Primer dijagrama arhitekture i infrastrukturnih dijagrama dizajniranih korišćenjem Power Designer alata



Slika 1.5 Primer arhitektonskog dijagrama sistema

PRIMER 3

Dijagram arhitekture iTunes-a



Slika 1.6

ZADATAK

Zadaci za proveru znanja

1. Nacrtati arhitekturu sistema za online naručivanje proizvoda
2. Nacrtajte dijagrame koji pokazaju koncepcijski i procesni pogled na arhitekturu sledećih sistema:
 - Sistema za automatsku prodaju železničkih karata na železničkoj stanici,
 - Kompjuterski upravljan sistem za video konferencije, a koji dozvoljava video, audio, i računarskih podataka dostupnim učesnicima video konferencije,
 - Robotizovani čitač podova. On mora da ima senzore za izbegavanje zidova i drugih prepreka.
 - Da li bi trebalo da postoji posebna profesija arhitekte softvera čija bi uloga bila da radi nezavisno sa klijentima radi projektovanja arhitekture? U tom slučaju, posebna kompanija bi mogla da primeni taj sistem. Koje bi se teškoće mogle javiti u primeni te profesije?

▼ Poglavlje 2

Odluke u vezi arhitekture

PITANJA NA KOJE ARHITEKTA SISTEMA TREBA DA NAĐE ODGOVORE

Projektovanje arhitekture softvera je kreativan proces u kome se projektuje organizacija sistema koja treba da zadovolji funkcionalne i ne-funkcionalne zahteve sistema.

Projektovanje arhitekture softvera je kreativan proces u kome se projektuje organizacija sistema koja treba da zadovolji funkcionalne i ne-funkcionalne zahteve sistema. Aktivnosti ovog procesa zavise od tipa sistema koji se razvija, od iskustva arhitekte sistema i od specifičnih zahteva sistema. To je u stvari proces donošenja odluka.

Arhitekte sistema treba da uzmu u razmatranje sledeća fundamentalna pitanja o sistemu:

1. Da li postoji opšta arhitektura sistema koja se može koristiti kao uzor za sistem koji se treba da projektuje?
2. Kako će sistem da se podeli po procesorima?
3. Koje arhitektonske mustre ili stilovi se mogu koristiti?
4. Koji će se fundamentalni pristup strukturiranju sistema primeniti?
5. Kako će se komponente strukture dalje podeliti u pod-komponente?
6. Koja će se strategija kontrole rada komponenti upotrebiti?
7. Koja je organizacija arhitekture najbolja za obezbeđenje nefunkcionalnih zahteva sistema?
8. Kako će se vrednovati projektno rešenje arhitekture?
9. Kako će se dokumentovati arhitektura sistema?

Iako je svaki softverski sistem jedinstven, sistemi u istom domenu primene često imaju slične arhitekture, u saglasnosti sa osnovnim konceptima domena. Zato se treba videti koje klase sistema i aplikacije imaju ta zajednička svojstva, i da se na osnovu toga doneše odluka koja se primenjena arhitektura može ponovo upotrebiti.

U slučaju ugrađenih sistema, ili sistema na personalnim računarima, obično se koristi samo jedan procesor, te se ne može projektovati distributivna arhitektura sistema, kao što se mora raditi kod velikih sistema, koji koriste više procesora. Odluka o primeni distribuirane arhitekture sistema je od ključne važnosti jer utiče na performanse i pouzdanost sistema.

Projektno rešenje arhitekture sistema može se zasnovati na nekoj posebnom šablonu arhitekture. Arhitektonski šabloni (engl., **architectural pattern**) su opisi organizacije sistema, kao što su arhitektura klijent-server, ili višeslojna arhitektura. Arhitektonski šabloni sadrže

bitna arhitektonska rešenja koja su upotrebljena kod različitih sistema. Preporučljivo je da se njihova eventualna upotreba razmotri na početku projektovanja arhitekture sistema.

NEFUNKCIONALNI ZAHTEVI I ARHITEKTURA SISTEMA

Pri projektovanju arhitekture, pojedini zahtevi vode ka konfliktnoj situaciju prilikom projektovanja arhitektura

Pri izboru stila arhitekture i strukture sistema, treba voditi računa o nefunkcionalnim zahtevima sistema:

1. *Performanse*: Ako su performanse bitne za sistem, onda arhitektura treba da obezbedi lokalizaciju kritičnih operacija unutra malog broja komponenti, u okviru istog računara. To znači da je bolje koristiti veliku umesto male komponente da bi se smanjila komunikacija između komponenti, koja odnosi dosta vremena.
2. *Bezbednost*: Ako je bezbednost bitna, višeslojna arhitektura je dobro rešenje, da bi se najkritičnije vrednosti stavile pod kontrolu u unutrašnjim slojevima sistema, sa visokom dozom zaštite.
3. *Zaštita*: Ako je zaštita od značaja, onda arhitektura treba da obezbedi da se zaštićene operacije obavljaju u samo jednoj komponenti ili u vrlo malom broju komponenti. To smanjuje troškove zaštite i primenu sistema zaštite koji mogu da isključe sistem u slučaju nekog kvara.
4. *Raspoloživost*: Ako je raspoloživost kritičan zahtev za sistema, onda arhitektura treba da bude tako projektovana da uključuje redundantne (ponovljive) komponente koje mogu jedna-drugu da zamene u u slučaju da jedna otkaže.
5. *Lakoća održavanja*: Ako je lakoća održavanja kritičan zahtev, arhitekturu sistema treba projektovati sa malim, samodovoljnim komponentama koje se lako zamjenjuju. Proizvodnja podataka se odvaja od korisnika a izbegava se deljivost struktura podataka.

Pri projektovanju arhitekture, pojedini zahtevi vode ka konfliktnoj situaciju prilikom projektovanja arhitektura. Na primer, velike komponente poboljšavaju performanse, a male olakšavaju održavanje. Ako su oba zahteva važna, mora se onda tražiti kompromisno rešenje. Ponekad, za pojedine delove sistema se onda, koriste različiti stilovi arhitekture.

Ocena arhitekture se može dobiti i njenim upoređenjem sa referentnim arhitekturama (ranije primjenjenim i provereno dobrom) ili sa opštim arhitektonskim šablonima (uzorima). Konačnu ocenu arhitekture daće ocena rada celog sistema, i zadovoljenje svih funkcionalnih i nefunkcionalnih zahteva.

PRIMER

Nefunkcionalni zahtevi i arhitektura sistema za kontrolu letova

Za sistem kontrole letova neophodno je voditi računa o performansama, sigurnosti i dostupnosti sistema. Kako sve ovo spada u nefunkcionalne zahteve, svaki ponaosob mora da bude ostvaren kroz odgovarajuću arhitekturu. U ovakvim sistemima, arhitektura mora da podrži svaki nefunkcionalni zahtev jer su oni kritični za rad i funkcionisanje samo sistema.

ZADATAK

Projektovanje arhitekture ssoftverskog sistema

1. Objasnite zašto se projektovanje arhitektura često radi i pre nego što se specifikacija zahteva kompetirana
2. Šta je posao arhitekte sistema?
3. Objasnite zašto može doći do konflikta pri projektovanju arhitekture sistema za koji su raspoloživost i sigurnost dva najvažnija nefunkcionalna zahteva.

✓ Poglavlje 3

Projektovanje arhitekture sistema

POGLEDI NA ARHITEKTURU SISTEMA

Svaki model pokazuje samo jedan pogled ili perspektivu sistema. Obično se koriste logički, procesni, razvojni i fizički pogled na arhitekturu sistema.

Svaki model pokazuje samo jedan pogled ili perspektivu sistema. Korisno je predstaviti više pogleda softverske arhitekture kada se različite informacije o sistemu traže u različitim vremenima (fazama razvoja). Krutchen (1995) je postavio svoj 4+1 pogled modela arhitekture softvera, sa četiri fundamentalna arhitektonska pogleda, uz pomoć slučajeva korišćenja i scenarija:

1. **Logičan pogled** - pokazuje ključne apstrakcije sistema kao objekte ili klase objekata. Zgodan je za povezivanje sistemskih zahteva sa ovim entitetima sistema.
2. **Procesni pogled** - pokazuje kako u fazi rada, sistem radi korišćenjem povezanih procesa. Koristan je za ocenu nefunkcionalnih karakteristika, kao što su performanse i raspoloživost.
3. **Razvojni pogled** - pokazuje dekompoziciju softvera radi njegovog razvoja, tj. prikazuje komponente koje su implementirane od strane razvojnog tima. Ovaj pogled je značajan za softverske menadžere i programere.
4. **Fizički pogled** - prikazuje hardverske i softverske komponente i njihovu distribuciju po procesorima sistema. Ovaj pogled je zgodan za planiranje razvoja i instaliranja sistema.

Pored ovih pogleda, koristi se i tzv. *Koncepcijski pogled, kao jedan apstraktan pogled na sistem koji može biti osnova za dekompoziciju uopštenih zahteva na detaljnije specifikacije zahteva. To pomaže inženjerima da odluče oko komponenti i njihovoј višestrukoj upotrebi, i da predstave jednu liniju proizvoda umesto jednog sistema.*

U praksi, koncepcijski pogledi se uvek razvijaju u toku procesa razvoja da bi se olakšalo donošenje odluka u vezi arhitekture. Oni olakšavaju komunikaciju sa akterima sistema. Ostali pogledi se razvijaju prema potrebi.

Pored neformalnih oznaka i načina prikazivanja arhitekture softvera, može se za te svrhe koristiti i UML, mada je on primereniji za faze projektovanja i implementacije sistema.

Inženjeri softvera koji primenjuju agilne metode osporavaju potrebu razvoja detaljne projektne dokumentacije, jer smatraju da se ona slabo koristi, a da njena priprema traži i vreme i novac. Međutim, nije sporno razviti pogledi na arhitekturu radi komunikacije sa drugim akterima sistema, bez potrebe da se pravi detaljna dokumentacija. Međutim, detaljna

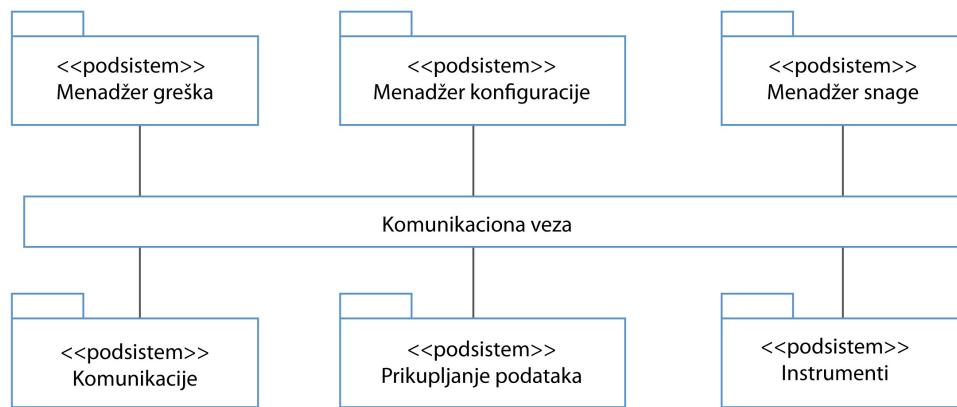
dokumentacija je potrebna kod kritičnih sistema, jer ona omogućava odobrenje projekta od strane regulacionih vlasti.

ARHITEKTURA SOFTVERA – PRIMER

Projektovanje arhitekture softvera je određivanje njegovih glavnih komponenti koje čine sistem i njihove međusobne interakcije.

Kada se odrede interakcije sistema koji se projektuje, i sistema i njegovom okruženju, prostupa se projektovanju arhitekture sistema. To znači da treba odrediti njegove glavne komponente koje čine sistem i njihove međusobne interakcije. Pri projektovanju arhitekture, koristi se neka od mustri za projektovanje arhitekture, te se one realizuje ili u vidi slojevite arhitekture, ili u vidu klijent-server modela, ili se primenjuje neka druga mustra.

Najapstraktniji nivo prikazivanja projekta arhitekture softvera stanice za prikupljanje vremenskih prilika je prikazan na slici1. Podsistemi sistema emituju poruke preko zajedničke komunikacione veze (magistrale). Svaki podsistem osluškuje poruke ove infrastrukture i preuzima poruke koje su mu namenjene.



Slika 3.1 Prikaz uprošćene arhitekture softvera stanice za prikupljanje vremenskih prilika

AKTIVNOSTI PROJEKTOVANJA SISTEMA

Projektovanje sistema definiše: ciljeve projektovanja, određuje arhitekturu softvera i njegove granične uslove

Model analize, dobijen analizom zahteva i mogućnosti, sadrži

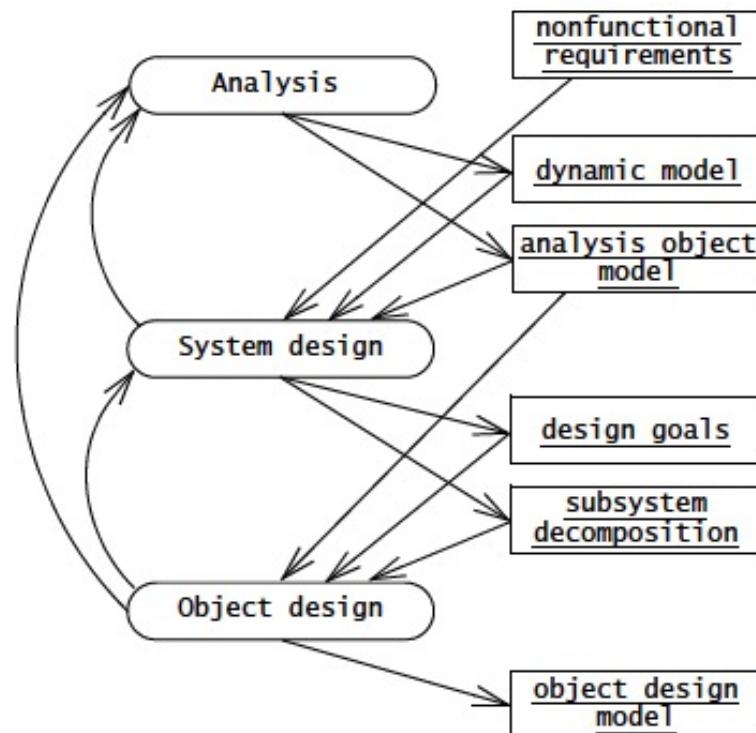
- skup nefunkcionalnih zahteva i ograničenja,
- model slučajeva korišćenja
- model objekata (Boundary, Contro i Entity)
- sekvencijalni dijagram ya svaki slučaj korišćenja, koji pokazuje redosled interakcije utvrđenih objekata.

Model analize je polazna osnova za projektovanje softverskog sistema. Model analize ne sadrži informaciju o unutrašnjoj strukturi sistema, njegovu konfiguraciju, a ni informaciju kako bi sistem realizovao postavljene zahteve.

Projektovanje softverskog sistema treba da dovede do sledećih rezultata:

- **ciljeve projektovanja**, koji opisuju kvalitet koji projektanti mora da ostvare,
- **arhitekturu softvera**, koja pokazuje kompoziciju sistema na podsisteme, sa njihovim vezama i odgovornostima, i njihovo mapiranje u hardver, kao i definisanje toka kontrole, pristupa i skladištenja podataka,
- **granične slučajeve korišćenja**, koji opisuju konfiguraciju, uključenje, isključenje i rad sa izuzecima.

Na slici 2 je prikazan dijagram aktivnosti projektovanja sistema, i njegovu povezanost sa ostalim aktivnostima softverskog inženjerstva.



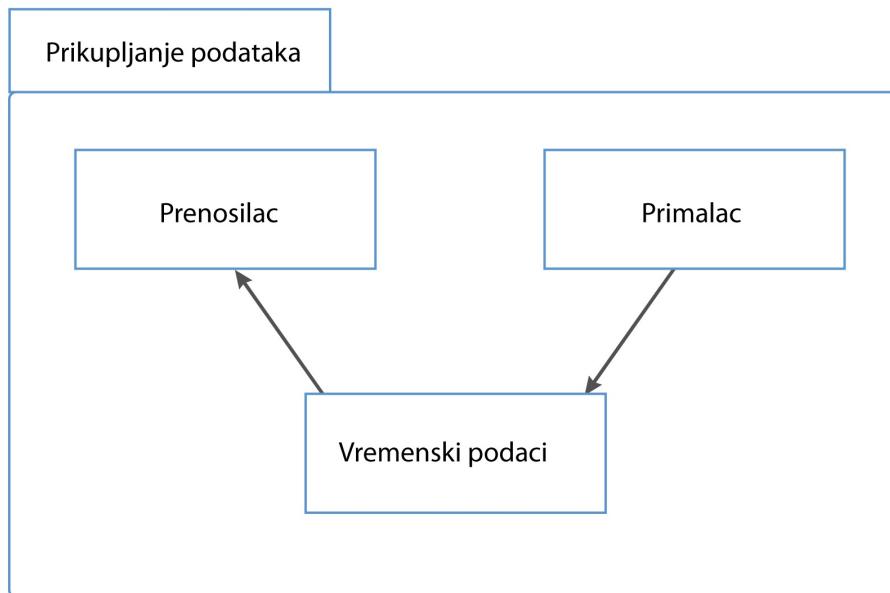
Slika 3.2 Aktivnosti projektovanja sistema

PODSISTEMI I NJIHOVA KOMUNIKACIJA

Kada komunikacioni sistem primi neku kontrolnu komandu, komandu dobija svaki od podistema, te svaki od njih samostalno reaguje na najbrži način

Kada komunikacioni sistem primi neku kontrolnu komandu, kao na primer, instrukciju za isključenje, komandu dobija svaki od podistema, te svaki od njih se isključuje samostalno na najbrži način. Kako sistem koji šalje poruku ne mora da zna adresu podistema kome šalje poruku, ova arhitektura sistema lako podržava različite konfiguracije sistema podistema sistema.

Na slici 5 prikazana je arhitektura podsistema za prikupljanje podataka, koji je deo cele arhitekture sistema na slici 1 . Pošiljalac i primalac, kao objekti, upravljaju komunikacije i objekt VremenskiPodaci sadrži informaciju prikupljenu od instrumenata i koja se prenosi informacionom sistemu za praćenje vremena. Ovakva struktura odgovara mustri proizvođač-korisnik.

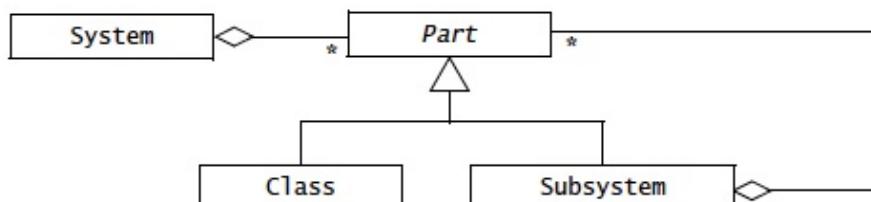


Slika 3.3 Arhitektura sistema za prikupljanje vremenskih podataka

PODSISTEMI I KLASE

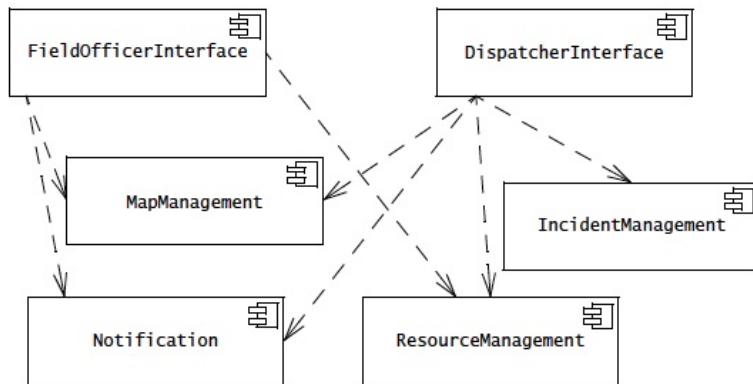
*Podsistemi su zamjenljivi deo sistema sa jasno definisanim interfejsima.
 Podsistemi sadrži klase.*

Podsistemi je zamjenljivi deo sistema sa dobro definisanim interfejsima koji učaruju stanja i ponašanja. Podelom sistema na podsisteme, omogućava se relativno nezavisno razvoj od strane različitih članova tima za razvoj. Dekompozicija sistema nije samo ograničena na podsisteme, jer se i ovi mogu dalje dekomponovati (slika 3).



Slika 3.4 Dekompozicija sistema

Na slici 4 prikazana je dekompozicija sistema za upravljanje incidentima.



Slika 3.5 Primer dekompozicije sistema za upravljanje rizicima

Java realizuje podsisteme primenom paketa (packages)

SERVISI

Servis je skup operacija koji dele neku zajedničku svrhu, a interfejsi podistema sadrže spisak servisa.

Karakteristično je za podsisteme je da daju određene servise drugim podsistemima.

Servis je skup operacija koje dele neku zajedničku svrhu. **Interfejs** podistema sadrži spisak operacija koje klase jednog podistema. On sadrži nazine operacija, njihove parametre, njihove tipove, i tip povratne vrednosti.

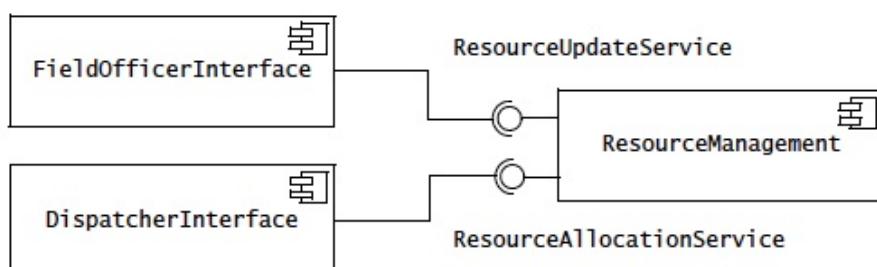
Projektovanjem sistema određuju se servisi podistema.

Projektovanje objekata (Object Design) se fokusira na programski interfejs aplikacije, tj. **API-Application Programmer Interface**, koji proširuje i dopunjuje interfejs podistema.

Za **povezivanje interfejsa**, UML koristi tzv. **konektori** (*assembly vonnetctors*). Nazaju se i soketima, tj. *ball and socket connectors* (slika 6).

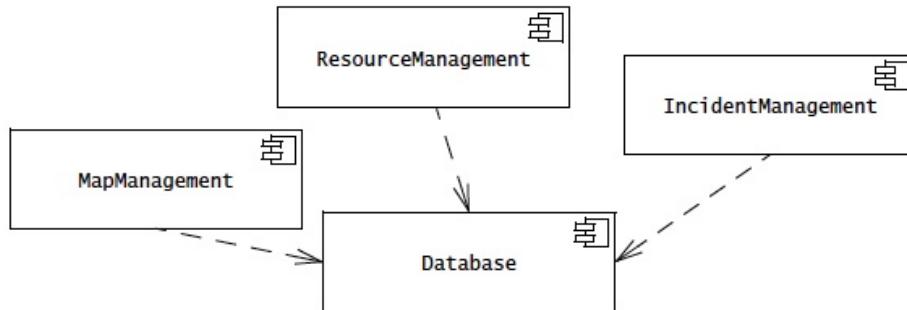
Na slici 4 je prikazan primer podistema za rad sa bazom podataka, sa svojim servisima, koji je dodat podsistemima na slici 7

Coupling (spojka) je broj zavisnosti (**dependencies**) između dva podistema. Podsistemi sa jakom povezanošću (coupling) se teže menjaju, jer promena kod jedno utiče i na drugi. Kod slabo povezanih podistem (loosely coupled) imaju veli stepen nezavisnosti..



Slika 3.6 Servisi podsistema

Tokom projektovanja, odlučeno je da se podsistemima na slici 3 podsistem koji bi ih povezivao sa bazom podataka (slika 5)



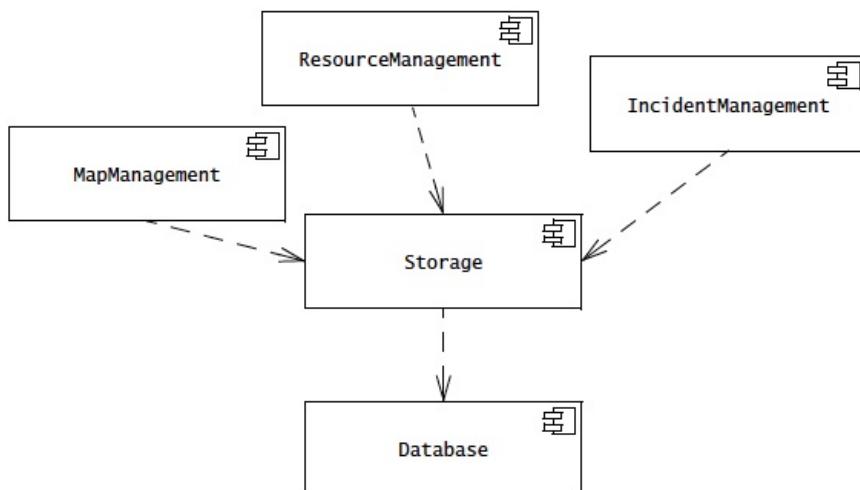
Slika 3.7 Podsistemi sa slike 3, prošireni sa podsistemom za rad sa bazom podataka

VEZA SA PODSISTEMIMA I KOHEZIJA PODSISTEMA

Potrebno je naći balans između veza podistema (coupling) i kohezija podistema.

Da bi podsistemi bili što nezavisniji od proizvođača sistema baza podataka (**DBMS - Database Management System**), može se između njih i sistema baze podataka, umetnuti novi podsistem koji svoje servise prema podsistemima ne menja, ali menja samo interfejs prema podsistemu baze podataka, tj. koristi onaj koji je specifičan za određeni sistem baza podataka, tj. koji yavisi od njegovog proizvođača.

Na slici 8 prikazan je taj slučaj, te podsistem Storage obavlja funkciju posrednika između podistema koji koriste usluge podistema rada sa bazom podataka i samog sistema baze podataka.



Slika 3.8 Podsistem Storage je neutralni posrednika između podistema i podistema sistema baze podataka.

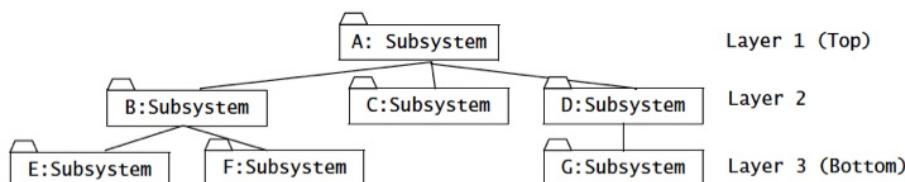
Kohezija (Cohesion) je broj zavisnosti unutar jednog podsistema. Cilj projektovanja je dobijanje podsistema sa visokom kohezijom.

Potrebno je nači balans između veza podsistema (coupling) i kohezija podsistema. Ako podsistemi imaju visoku kohiziju, obično imaju nisku međusovnu povezanost (mali coupling). To dovodi do povećanja broja podsistema, ali i do povećanja ukupnog broja interfeksa među podsistemima. Smatra se da je kompromis imati 7 +/- 2 koncepta na istom nivou apstrakcije. To ynači da ako ima više od 9 podsistema, tj. da jedan podsistem mora da ima više od 9 servisa, preporuka je da se izvrši revizija podsistema, d abi se taj broj veza i servisa smanjio.

PRIMENA SLOJEVA

Jedna sloj predstavlja grupu podsistema koji daju povezane servise, pri čemu koriste i servise sljeva ispod njih

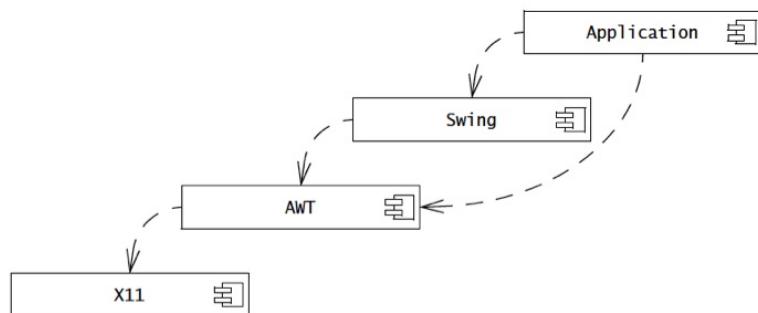
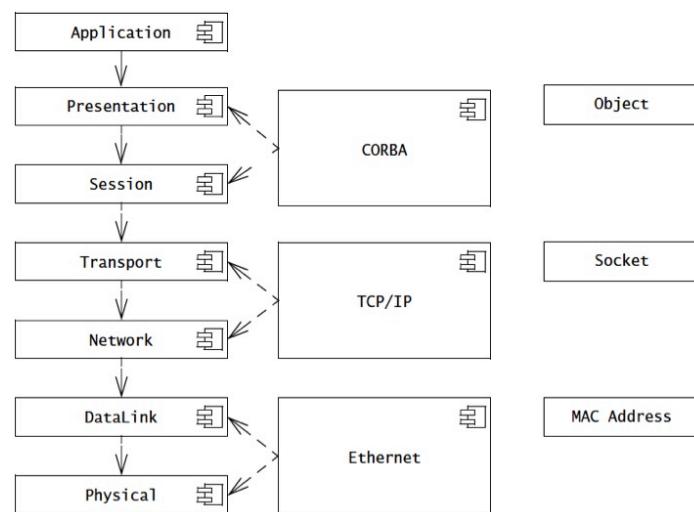
Hijerarhijska dekompozicija sistema dovodi do **slojevite arhitekture** sistema (slika 9). Jedna **sloj** predstavlja grupu podsistema koji daju povezane servise, pri čemu koriste i servise sljeva ispod njih i nema saznanja o slojevima iznad njega. Slojevi se redaju tako, da zavise samo od slojeva ispod njih.a najniži sloj ne zavisi ni od jenog sloja. Sloj na vrhu ne daje servise nijednom drugom



Slika 3.9 Slojevita arhitektura sistem

Na slici 10 dat je primer slojevite arhitekture sistema koji obezbeđuje servise za rad distribuiranih sistema na Internetu

Na slici 11 prikazan je primer otvorene slojevite arhitekture. Ona dovoljava da viši sloj preskače neki niži sloj i pristupa direktno nekom još nižem sloju. Na taj način se može ubrzati rad softverskog sistema, ali se gubi na fleksibilnosti i nezavisnosti slojeva.



Slika 3.10 Primer otvorene slojive arhitekture

ZADATAK

Projektovanje arhitekture softverskog sistema

1. Dajte predlog arhitekture sistema banke primenom slojeva

▼ Poglavlje 4

Arhitektonski šabloni

ŠTA SU ARHITEKTONSKI ŠABLONI?

Arhitektonski šabloni opisuju organizaciju sistema koja se pokazala uspešnom u ranijim sistemima

Primena arhitektonskih šablona je način predstavljanja, deljenja i zajedničke upotrebe znanja o softverskim sistemima. Šabloni su apstraktni opisi dobre prakse, koja je testirana i oprobana u različitim sistemima i okruženjima. Arhitektonski šabloni opisuju organizaciju sistema koja se pokazala uspešnom u ranijim sistemima. Svaki šablon ima svoju oblast primene, kao i slabosti i dobre osobine. To isto treba da bude dokumentovano za svaki šablon.

ZADATAK

Odgovorite na sledeća pitanja

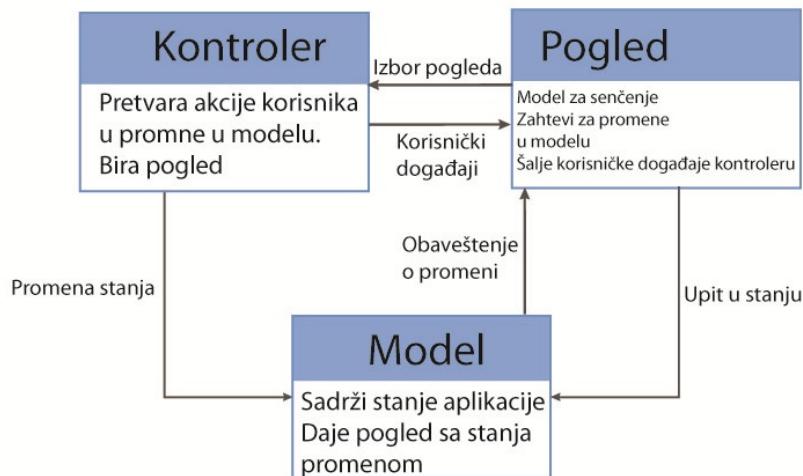
1. Čemu služe arhitektonski šabloni?
2. Pokušajte na internetu da pronađete spisak arhitektonskih šablona koji se koriste.

▼ 4.1 MVC arhitektura

MVC ILI MODEL-POGLED-KONTROLER ŠABLON

Arhitektonski šabloni opisuju organizaciju sistema koja se pokazala uspešnom u ranijim sistemima

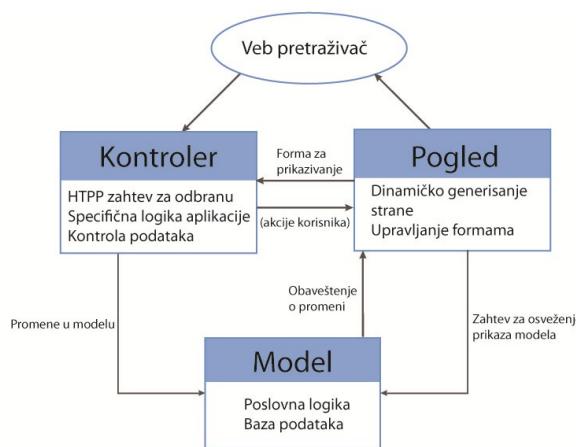
Radi ilustracije, na slici 1 je prikazan dobro poznati šablon **Model-Pogled-Kontroler** (engl. **Model-View-Controller**), ili MVC šablon koja se koristi kod mnogih veb sistema.



Slika 4.1.1 Koncepcijски pogled на MVC шаблон

Pored grafičkog modela, opis šablonu obezbeđuje njeno ime, opis upotrebe i primer tipa sistema u kome je šablon применjen (Tabela 1). Takođe, daje se i preporuka gde se šablon može koristiti, i navode se njegove prednosti i nedostaci..

Ime	MVC
Opis	Odvaja prezentaciju i interakciju od podataka sistema. Sistem se deli na tri logične komponente koje su u interakciji. Komponenta Model upravlja podacima sistema i operacijama nad podacima. Komponenta Pogled (View) definiše i upravlja prezentacijom podataka koje korisnik vidi. Komponenta Kontroler (Controller) upravlja interakcijom korisnika i prenosi te interakcije na Pogled i Model
Primer	Veb aplikacija prikazana na slici 3
Kada se koristi	Upotrebljava se kada postoji više načina pogleda i interakcije sa podacima. Takođe, upotrebljava se i kada budući zahtevi za interakcijom i za prezentacijom po podatkovima nisu poznati unapred.
Prednosti	Omogućava promenu podataka nezavisno o njihove prezentacije, i suprotno. Omogućava da se isti podaci predstave na različite načine, a da se pri promeni podataka ta promena vidi na svim različitim prezentacijama.
Nedostaci	Može zahtevati dodatni kod i uvećava složenost koda i u slučaju primene kod jednostavnih modela.



Slika 4.1.2 Izvršni pogled на MVC шаблон

MVC (MODEL-VIEW-CONTROL) ARHITEKTURA

MVC arhitektura scvrstava sve podsistema u tri kategorije:Model, View i Controler

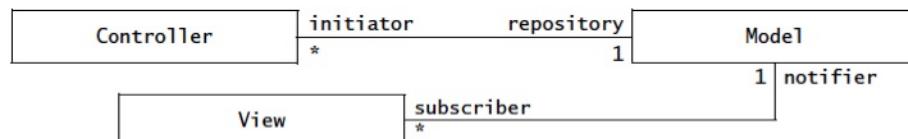
MVC arhitektura (slika 4) ima podsisteme svrstane u tri tipa:

- **model podsistem** . održava domensko znanje
- **view podsistem** - prikazuje model korisniku
- **controler podsistemi**- upravljaju redosledom interakcija sa korisnikom.

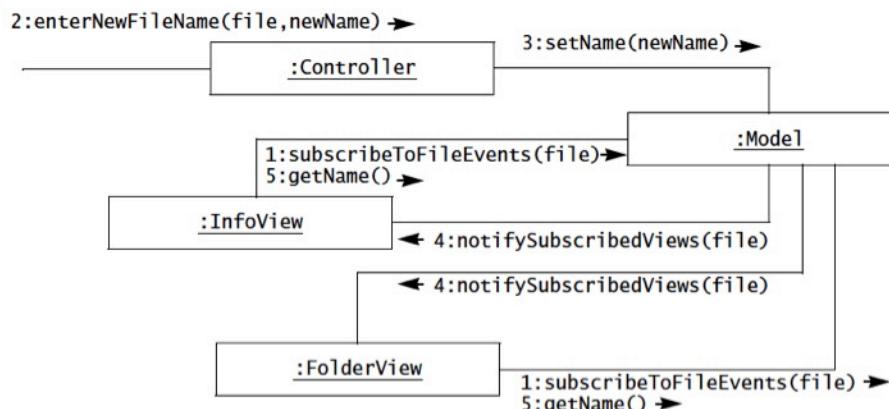
Model podsistem je tako razvijen da ne zavisi od view i controler podsistema.Promene svog stanja prenosi na view podsistem primenom subscribe/notify protokola

MVC je je specijalni slučaj repozitorijuma gde Model primenjuje centralnu strukturu podataka, a controlni objekti kontrolišu tok kontrolnih poruka.

Na slici 5 prikazan je primer komunikacionog dijagrama sistema koji primenjuje MVC arhitekturu, a koji prikazuje sekvene događaja.



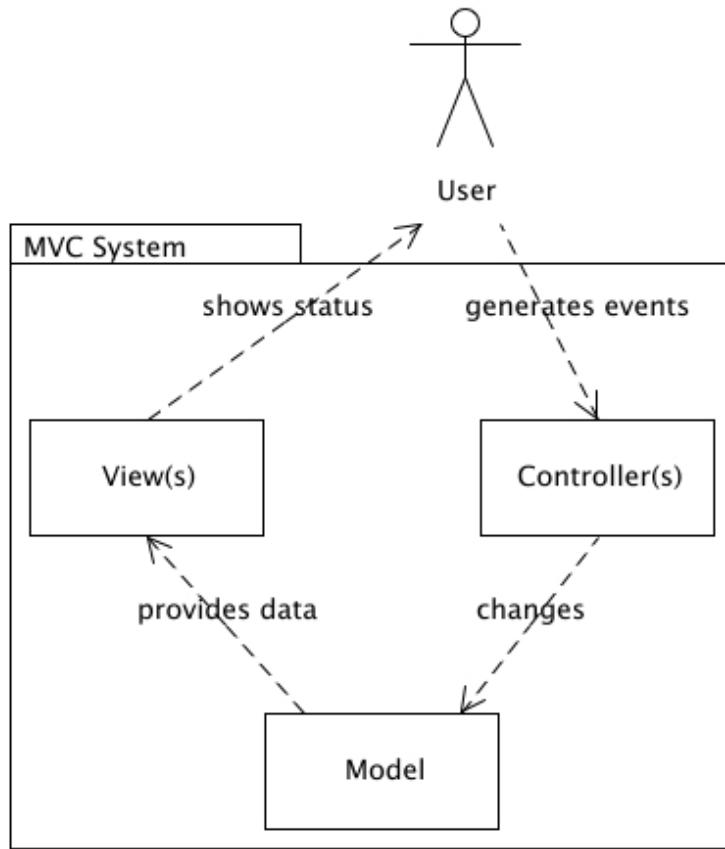
Slika 4.1.3 MVC arhitektura



Slika 4.1.4 Primer toka događaja u sistemu sa MVC arhitekturom.

PRIMER

MVC arhitektura



Slika 4.1.5 Primer MVC arhitekture aplikacije za planiranje događaja

ZADATAK

Proverite svoje razumevanje MVC arhitekture

1. Na primeru aplikacije rezervacije avionskih karata prikazati MVC šablon,

❖ 4.2 Slojevita arhitektura

ARHITEKTURA IZRAŽENA U FUNKCIONALNIM SLOJEVIMA

Slojevita arhitektura obezbeđuje odvajanje i nezavisnost komponenata softvera primenom funkcionalnih slojeva. Svaki sloj zavisi od uređaja i servisa koje nudi sloj ispod njega

Slojevita arhitektura obezbeđuje odvajanje i nezavisnost komponenata softvera primenom funkcionalnih slojeva (Tabela 1). Svaki sloj zavisi od uređaja i servisa koje nudi sloj ispod njega.

Ime	Slojevita arhitektura
Opis	Organizuje sistem u vidu slojeva koji se odnose na funkcionalnost specifičnu za svaki sloj. Sloj obezbeđuje servise sloju iznad njega, tako da najniži sloj obezbeđuje ključne servise koji se najčešće koriste u svim slojevima.
Primer	Model slojevite arhitekture sistema koji obezbeđuje zajedničko korišćenje dokumenata koji se drže u različitim bibliotekama, kao što je prikazano na slici 2.
Kada se upotrebljava	Kada se dodaju novi uređaji i servisi, s anovom funkcionalnožu, preko postojećih sistema, Kada se razvoj radi preko nekoliko timova pri čemu je svaki tim odgovoran za razvoj funkcionalnosti jednog sloja. Kada postoji zahtev za primenu višeslojne bezbednosti.
Prednosti	Omogućava zamenu celog sloja, uz uslov da s etime na menja interfejs. Ponovljive funkcije (npr. autentikacija) se mogu obezrediti u svakom sloju radi povećanje nezavisnosti od sistema.
Nedostaci	U praksi, teško je ostvariti jasnu podелу između slojeva a najviš sloj često ima komunikaciju sa nižim slojevima, a ne samo sa slojem ispod njega. Performanse mogu biti problem jer se u svakom sloju mora da vrši interpretacija zahteva za servisom, tj. obrada u svakom sloju.

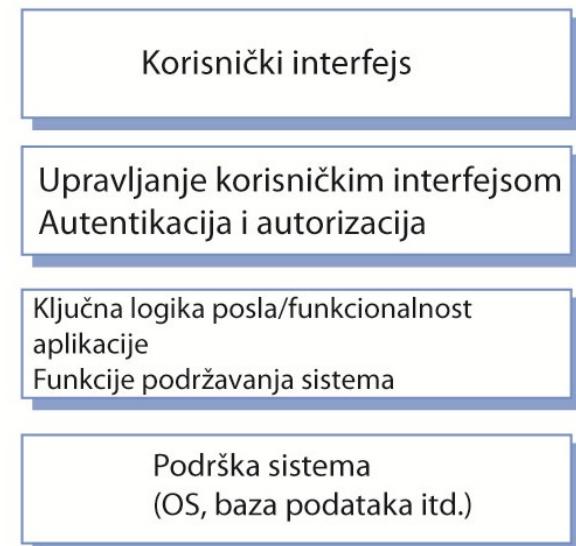
Primena slojevite arhitekture softverskog sistema omogućava njegov inkrementalni (postepeni) razvoj. Čim se razvije jedan sloj, njegovi servisi se mogu koristiti, nezavisno od kasnijih servisa novih slojeva, u nadgradnji sistema. Arhitektura se lako menja i prenosi. Važno je obezrediti da se interfejsi ne menjaju jedan sloj se može zameniti drugim, ekvivalentnim slojem (obavlja istu funkciju, ali na drugi način). Ako se promeni funkcionalnost jednog sloja, to se održava samo na sloj iznad njega.

Slojeviti sistemi lokalizuju zavisnost od mašine (npr. operativnog sistema, tipa baze podataka) samo na unutrašnje (donje) slojeve. To olakšava kreiranje multi-platformske implementacije neke aplikacije. U tom slučaju, za svaku mašinu se menjaju samo ti unutrašnji slojevi

PRIMERI ARHITEKTURE SA SLOJEVIMA

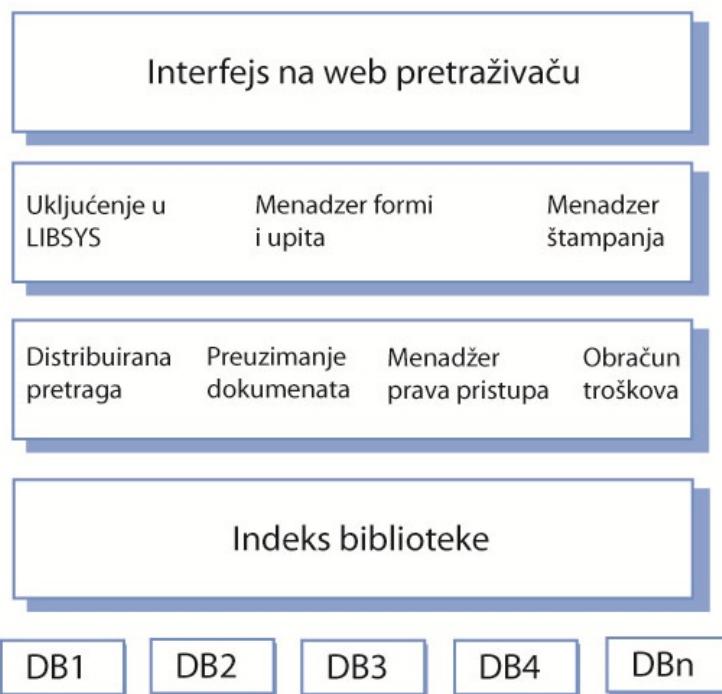
Svaki sloj obezbeđuje novu funkcionalnost, a oslanja se na funkcionalnost slojeva ispod njega.

Na slici 1 prikazan je primer arhitektura sistema sa četiri sloja. Najniži sloj obezbeđuje podršku podržavajućeg sistema, kao na primer, određenog sistema baze podataka (DBMS) ili određenog operativnog sistema. Sledeći sloj je aplikacioni sloj koji sadrži komponente koje obezbeđuju funkcionalnost aplikacije i korisničke komponente, koje koriste druge komponente aplikacije. Treći sloj obezbeđuje upravljanje korisničkim interfejsom, kao i autentikaciju (identifikaciju) i autorizaciju (ovlašćenje) korisnika. Četvrti, najviši sloj, implementira korisnički interfejs. Ovi slojevi se mogu dalje deliti na dva ili više slojeva.



Slika 4.2.1 Grafički prikaz slojevite arhitekture softverskog sistema

Na slici 2 prikazan je primer primene šabloni višeslojne arhitekture bibliotečkog softverskog sistema, LIBSYS, koji omogućava elektronski pristup zaštićenom materijalu iz grupe univerzitetskih biblioteka. On koristi pet slojeva, pri čemu u je najniži sloj zavistan od sistema baze podataka koji koristi svaka biblioteka.



Slika 4.2.2 Arhitektura LYBSYS sistema

ZADATAK

Arhitektura informacionih sistema

1. Upotrebom osnovnog modela arhitekture informacionih sistema, datom na predavanju, predložite komponente koje bi mogle biti deo informacionog sistema koji bidozvolio korisnicima da vide informacije o dolascima i odlascima aviona na određenom aerodromu.

▼ 4.3 Arhitektura sa skladištem podataka

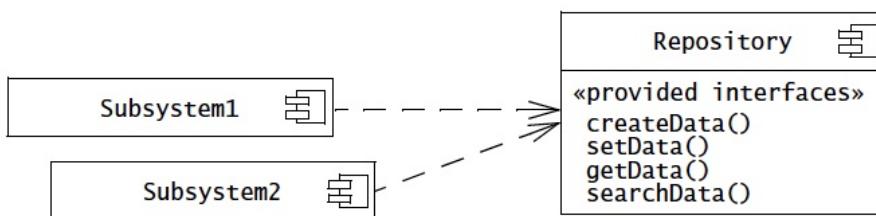
REPOZITORIJUM - ZAJEDNIČKO SKLADIŠTE PODATAKA

Repozitorijum obezbeđuje pristup svih podistema jedinstvenoj strukturi podataka.

Arhitektura softvera uključuje dekompoziciju sistema, globalni tok kontrole, rad sa graničnim uslovima, i protokole komunikacije između podistema.

Primenom različitih arhitektonskih stilova, oklašan je i ubrzan rad projektanata, jer pri izboru određenog arhitektonskog stila, oni preuzimaju sva svojstva izabrane arhitekture, te nije potrebno da to samo definišu.

U slučaju arhitektonskog stila **Repozitorijum**, podsistemi pristupaju jednom jedinstvenoj strukturi podataka, koji se naziva **centralni repozitorijum** (skladište podataka). (slika 1).



Slika 4.3.1 Repozitorijum, kao centralno skladište trajnih podataka

Repozitorijumi su tipični za aplikacije koji koriste sisteme baza podataka, Centralizacijom podataka, lakše se kontroliše pristup podacima i uređuje istovremeni rad korisnika s apodatacima. Takođe, lakše se kontroliše integritet povezanih podistema.

Repozitorijume treba koristiti kod aplikacija koje se stalno menjaju, a imaju složene zadatke obrade podataka. Kada je centralni repozitorijum dobro definisan, lako se mogu dodavati ili menjati servisi koji obezbeđuju podistemi. Glavni nedostatak primene repozitorijama je opasnost da daoni postanu "usko grlo" jer svi podistemi koriste isti repozitorijum.

Ako je veza repozitorijuma sa podistemima jaka, onda se oni teže menjaju, jer te promene zahtevaju i promene u repozitorijumu, a to pak, zahteva promene i u ostalim podistemima.

ARHITEKTURA KORIŠĆENJA ZAJEDNIČKOG SKLADIŠTA PODATAKA

Organizacija alata oko zajedničke baze podataka je način da se poveća efikasnost rada sa podacima, jer nema prenosa podataka sa jednog na druge module

Šablon za projektaovanje pod nazivom „Arhitektura sa skladištem podataka“ određuje skup komunicirajućih komponenti koje koriste i dele isto skladište podataka (slika 2).

Ime	Skladište podataka
Opis	Svi podaci sistema se upravljaju u centralnom skladištu kome mogu prići sve komponente sistema. Komponente ne komuniciraju direktno sa podacima, već preko skladišta (softverskog sloja koji upravlja podacima u bazi podataka).
Primer	Slika 2 prikazuje primer IDE sistema u kome sve komponente koriste skladište sistema sa informacijama od projektnom rešenju. Svaki softverski alat generiše informaciju koja je onda raspoloživa drugim alatima.
Kada se upotrebljava	Ova se mustra koristi u slučaju sistema sa velikom količinom informacija koje se moraju uskladištiti za dugi niz godina. Može se koristiti i kod sistema vođenim podacima u kojima smeštaj nekog podatka u skladište može da pokrene neku akciju ili da aktivira neki alat.
Prednosti	Komponente mogu biti nezavisne, tj. ne moraju da znaju postojanje drugih komponenti. Promene koje je izvršila jedna komponenta (u podacima), šire se svim drugim komponentama. Svi podaci se konsistentno upravljaju (npr. bekap se radi istovremen) kao da su na istom mestu.
Nedostaci	Skladište je jedinstvena tačka otkaza jer njenim otkazivanje, staje ceo sistem. U slučaju gусте komunikacije, performanse sistema mogu biti lošije. Teško je primeniti distribuisane baze skladišta podataka.

Slika 4.3.2 Opis šablon sa zajedničkim skladištem podataka

Najveći broj sistema koji se koriste veliku količinu podataka organizuju oko velike zajedničke baze podataka, tj. skladišta (eng., repository). Ovaj šablon projektovanja sistema je pogodna za takve slučajeve u kojima jedne komponente pune bazu podataka, a druga grupa ih koristi. Tako rade informacioni sistemi za upravljanje poslovanjem organizacija, CAD sistemi i interaktivni sistemi za razvoj softvera.

PRIMER ARHITEKTURE SA SKLADIŠTEM - IDE SISTEM

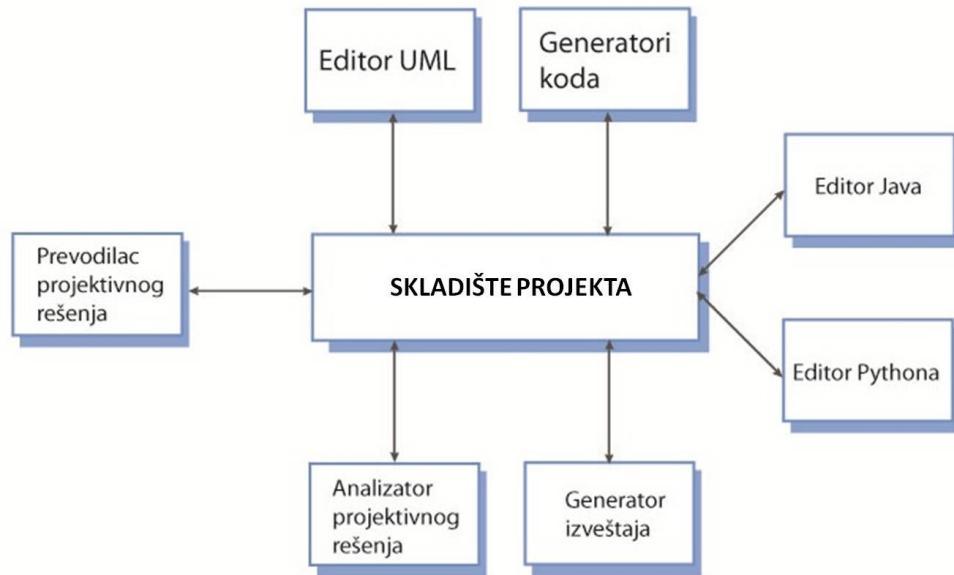
Kontraciom svih podataka na jednom mestu dobija se na efikasnosti sistema

Na slici 2 je prikazan jedan IDE sistem (Integrated Development Environment) koji koristi zajedničko skladište podataka.

Organizacija alata oko zajedničke baze podataka je način da se poveća efikasnost rada sa podacima, jer nema prenosa podataka sa jednog na druge module. Kako sve komponente koriste isti model podataka, otežano je dodavanje novih komponenata jer, po pravilu, one imaju svoje specifične modele podataka.

U sistemu na slici 3 , skladište je pasivno, jer rad sa podacima zavisi od rada komponenata.

To su „pasivna“ skladišta, za razliku od „aktivnih“, koji mogu da podstaknu aktivnost komponenata, kada se dogodi neki unapred definisan događaj..



Slika 4.3.3 Arhitektura sa skladištem (repozitorijem) jednog IDE sistema

ZADATAK

Odgovorite na sledeća pitanja

1. Navedite i nacrtajte arhitekturu sistema po izboru primenom repozitorijum šablonu.

▼ 4.4 Klijent-server arhitektura

KLIJENT/SERVER ARHITEKTURA

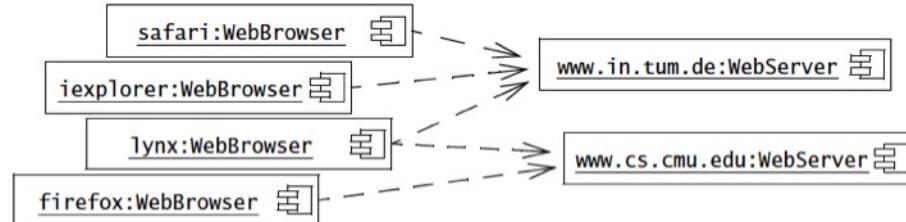
Serverski podsistem obezbeđuje servise klijentima, tj. drugim podsistemima.

Kod primene **Klijent/Server** arhitekture, serverski podsistem obezbeđuje servise instancama drugih podistema, koji se nazivaju klijentima, koji su odgovorni za interakciju sa korisnikom. Zahtev za servisom je obično iniciran sa udaljenom procedurom pozivanja (RPC) ili primenom zahedničkih objektnih brokera (npr. CORBA, Java RMI ili HTTP). Kontrolni tok kod klijenata i servisa je nezavisan sem u slučaju sinhronizacije upravljanja zahtevima ili prijemom rezultata.

Klijent/Server arhitektura je dobra kod distribuiranih sistema koji rade sa velikom količinom podataka. (slika 1,2)



Slika 4.4.1 Klijent-Server arhitektura



Slika 4.4.2 Primer Veb klijent-server arhitekture.

ŠABLON KLIJENT-SERVER ARHITEKTURE

Serveri ne moraju da znaju identitet klijenata, dok klijenti mogu da znaju nazine servera i njihovih servisa. Klijenti koristi servise upotrebom poziva udaljenih procedura kao što je HTP

Šablon klijent-server arhitekture je organizovan u vidu servisa i odgovarajućih servera, kao i klijenata koje pristupaju sistemu radi korišćenja tih servisa (Tabela 1).

Ime	Klijent-server arhitekture
Opis	Kod klijent-serverskih arhitektura funkcionalnost sistema je organizovana da pruža servise, gde svaki servis obezbeđuje poseban (softverski) server. Klijenti su korisnici ovih servisa.
Primer	Slika 2 prikazuje primer video/DVD biblioteke organizovane u vidu klijent-sever arhitekture.
Kada se upotrebljava	Upotrebljava se kada se podacima u bati podataka pristupa sa raznih lokacija. Kako se serveri mogu umnožavati, može se se broj servera povećavati u skladu sa opterećenjem sistema.
Prednosti	Glavna prednost ovog modela je mogućnost distribuiranja servera na mreži. Opšta funkcionalnost (npr. štampanje) se može nuditi svim klijentima, a ne mora da se na taj način nude svi servisi svima.
Nedostaci	Svaki servis je jedinstvena tačka otkaza, te može biti cilj napada na servise ili na server. Performanse je teško predvideti jer zavise od mreže i od sistema. Mogu se javiti i problemi upravljanja, ako su serveri pod kontrolom različitih organizacija.

Tabela-1: Opis šablon klijent-server arhitekture

Glavne komponente klijent-server modela su:

1. Skup servera koji nude servise drugim komponenata (pr., servisa za štampanje, fajl serveri, kompjajler serveri).
2. Skup klijenata koji pozivaju servise koje nude servere.

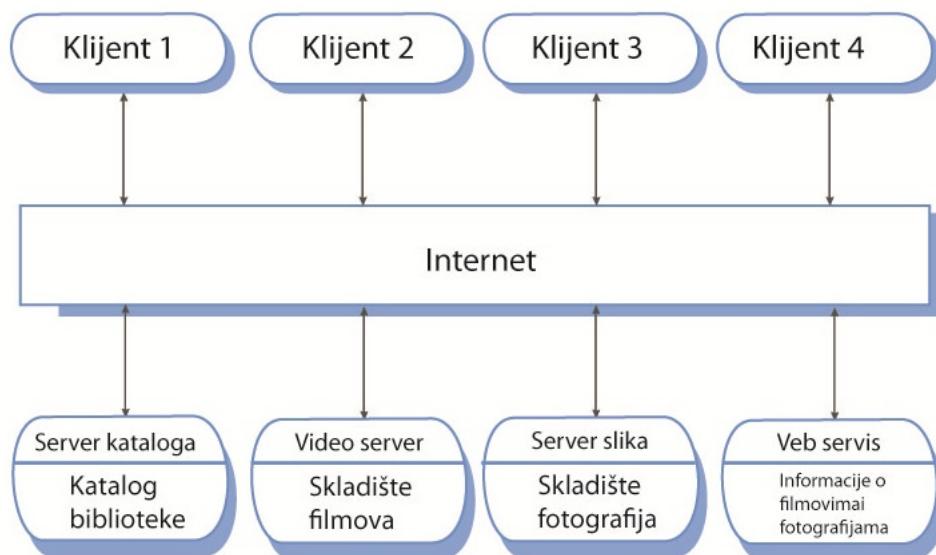
3. Mreža koja dozvoljava klijentima da pristupe servisima. Najčešće se klijent-server arhitektura realizuje u vidu distribuiranog sistema na mreži sa Internet protokolom.

Serveri ne moraju da znaju identitet klijenata, dok klijent mogu da znaju nazive servera i njihovih servisa. Klijenti koristi servise upotrebom poziva udaljenih procedura (engl. remote procedure calls-RPC), kao što je HTP . Klijent šalje poziv i čeka odgovor servera. Glavna prednost klijent-server modela je njegova distribuirana arhitektura. Mogu se koristiti i distribuirani procesori. U sistem se lako, po potrebi, mogu dodavati novi serveri, bez isključivanja sistema.

PRIMER KLIJENT-SERVER ARHITEKTURE

Glavna prednost klijent-server modela je njegova distribuirana arhitektura

Slika 3 prikazuje primer sistema koji koristi klijent-server arhitekturu. To je višekoristički veb sistem koji obezbeđuje biblioteku file-ova i fotografija. Video server radi kompresije i dekompresije raznih formata video zapisa (file-ova), da bi se ubrzao njih transfer mrežom. različitih upita i podržava prodaju fotografija..



Slika 4.4.3 Primer klijent-server arhitekture

ZADACI

Klijent-server arhitektura

1. Nacrtajte arhitekturu sistema online biblioteke primenom klijent-server šablonu
2. Nacrtajte arhitekturu elektronskog dnevnika učenika primenom klijent-server šablonu.

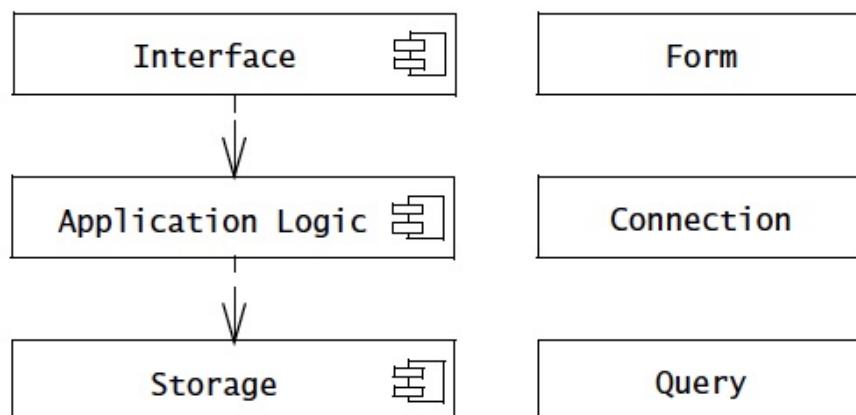
❖ 4.5 Troslojna i četvoroslojna arhitektura

ARHITEKTURA SA TRI I SA ČETIRI SLOJA

Tri sloja: interfejs (Boundary klase), sloj logike (Comtrol klase) i memorijski sloj (Entity klase)

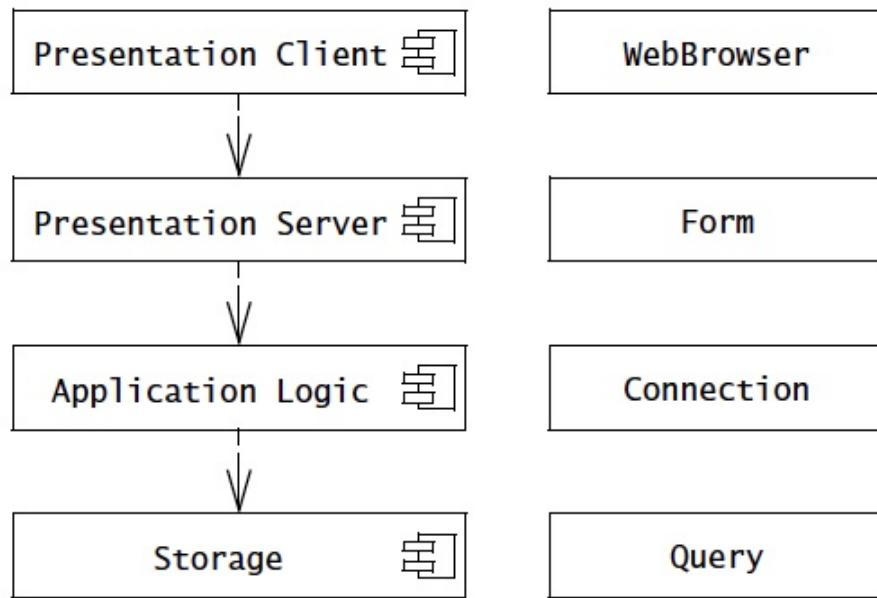
Troslojna arhitektura organizuje podsisteme u tri sloja:

- **interfejs sloj** - uključuje sve granične objekte (Boundary)
- **sloj logike aplikacije** - obuhvata sve kontrolne i entitetske objekte (Control i Entity) koji vrše obradu podataka, proveru pravila, i slanje obaveštenja u skladu sa potrebama aplikacije
- **memorijski sloj** - memorije, pretražuje i manipuliše sa trajno zapisnim podacima.



Slika 4.5.1 Three-teer (troslojna) arhitektura

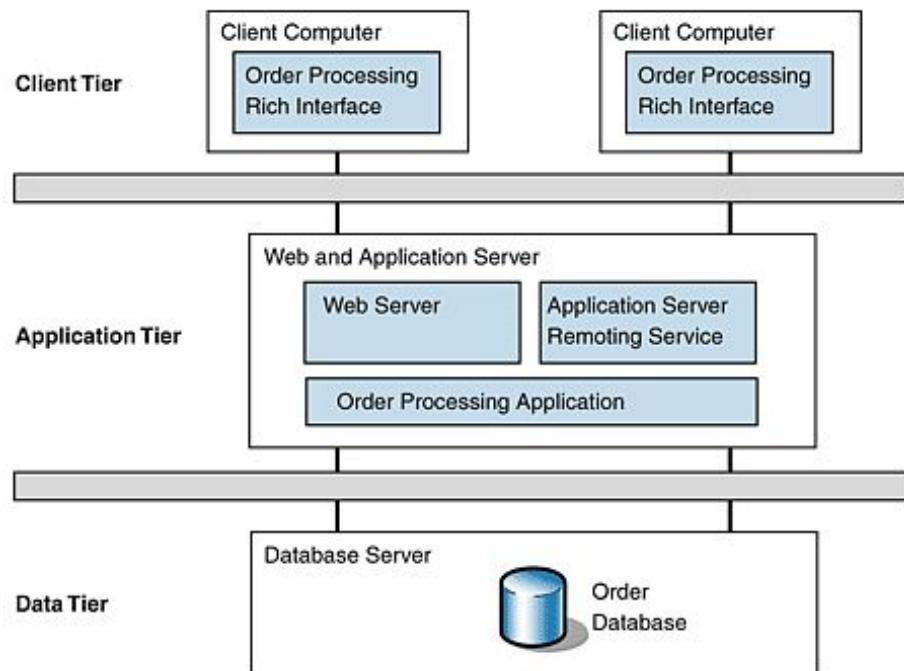
Četvoroslojna arhitektura (slika 10) je troslojna arhitektura u kojoj je interfejs sloj podeljen na interfejs na klijentu i interfejs na serveru (slika 10).



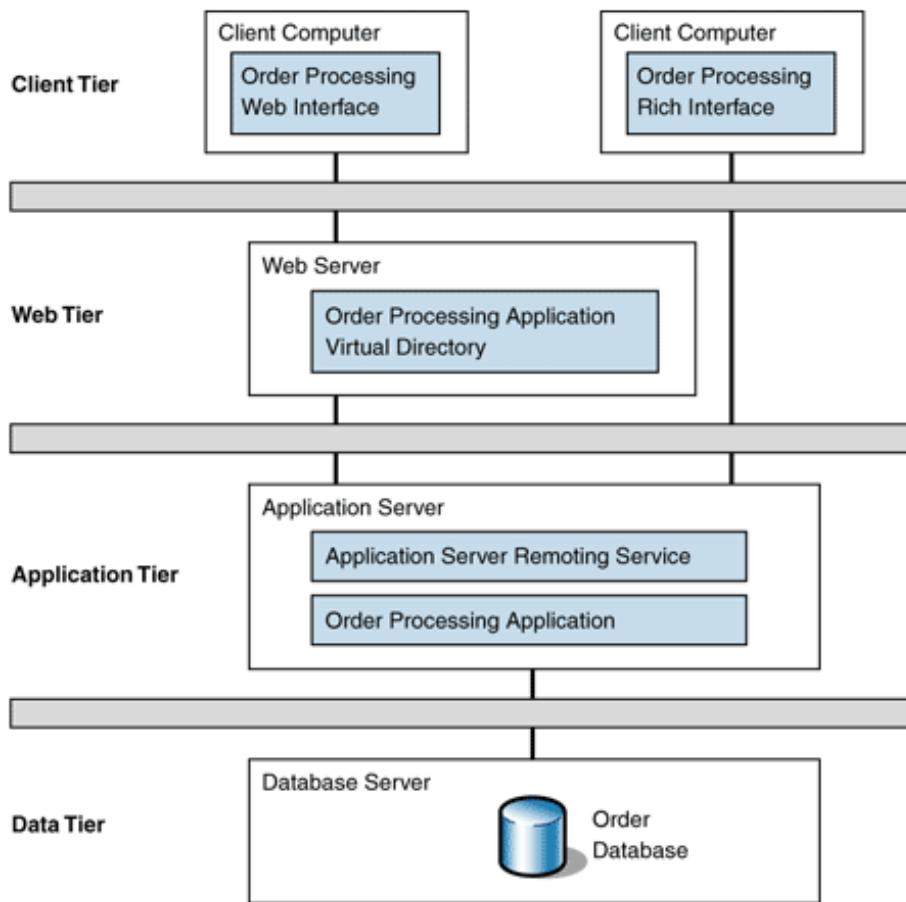
Slika 4.5.2 Four-teer (četvoroslojna arhitektura)

PRIMER

Primeri troslojne i četvoroslojne arhitekture



Slika 4.5.3



Slika 4.5.4

ZADACI

Proverite razumevanje troslojne i četvoroslojne arhitekture

1. Primere iz zadatka 8 modifikovati da primenjuju troslojnu i četvoroslojnu arhitekturu.

✓ 4.6 Servisno-orientisana arhitektura

ŠABLON SERVISNO-ORIJENTISANE ARHITEKTURA

Servisno-orientisana arhitektura organizuje aplikaciju kao kolekciju servisa koji komuniciraju jedan sa drugim preko dobro-definisanih interfejsa. Ovi servisi se nazivaju Veb servisima.

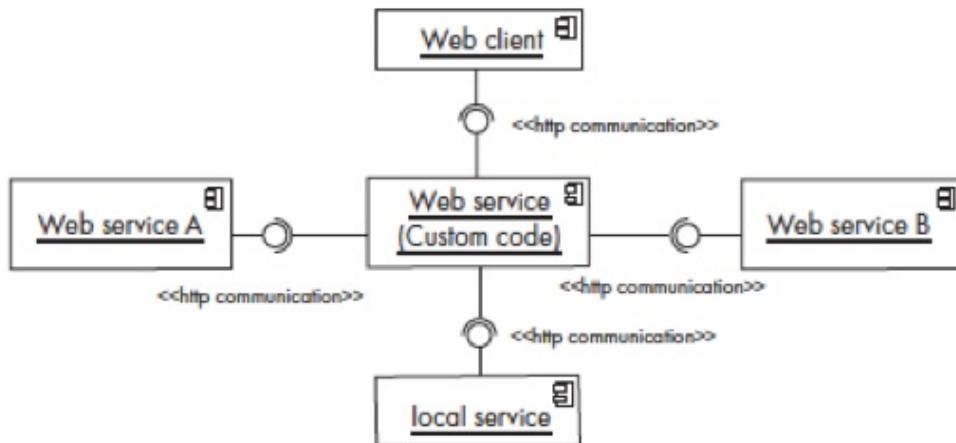
Servisno-orientisana arhitektura organizuje aplikaciju kao kolekciju servisa koji komuniciraju jedan sa drugim prekodobro-definisanih interfejsa. Ovi servisi se nazivaju Veb servisima.

Veb servis je aplikacija koja je dostupna preko interneta a koja može da bude integrisana sa drugim veb servisima i na taj način, zajedno, čin jednu Veb aplikaciju. Da bi se veb servis koristio, morate da mu pošaljete yahtev tačno formatiran u HTTP u njegov HTTP server. U ovom slučaju operacija se obavlja mimo veb pretraživača, u pozadini. Server izvršava servisnu aplikaciju i vraća rezultat u vidu dokumenta, tipično, u strukturisanom jeyaku, kao što je XML. Ovo je pokazano na slici 1.

Klijentski program koji razvijamo, dobija informaciju od Veb servisa koji obezbeđuje neka kompanija preko Interneta. To izaziva slanje

Klijentski program koristi protokole Veb servisa da bi pristupio lokalnom serveru koji radi u sitoj kompaniji.

Ključni aspekt arhitektura Veb servisa je da svi podsistemi međusobno komuniciraju primenom otvorenih Wew standarda. Sve komponente sistema su povezane preko Interneta, bez obzira gde se nalazile. Zato, Veb servis može da se koristi od strane mnogih aplikacija se mnogih lokacija.



Slika 4.6.1 Servisno-orientisana arhitektura

ZAŠTO KORISTITI VEB SERVISE?

Moraju se zadovoljiti dva zahteva: zaštita korisnika i informacija, kao i pouzdanost u radu.

Veb servisi mogu da izvrše široki opseg zadataka, počev od obrade jednostavnih zahteva, do vrlo složenih poslovnih obrada. Organizacije mogu da koriste Veb servise da bi automatizovale i poboljšale svoje operacije. Na primer, aplikacije elektronsog posovanja mogu koristiti veb servise da:

- pristupe bazama podataka o proizvodima svojih dobavljača,
- obrađuju kreditne kartice upotrebom Veb servisa banaka,
- organizuju isporuku upotrebom Veb servisa transportne organizacije.

Najveći izazov za inženjere softvera koji razvijaju Veb servise je sigurnost. Servis koji nudite, otvara vaše poslovne aplikacije i podatke vašim udaljenim klijentima. Zato, mora da se

posebna pažnja posveti zaštiti i korisnika servisa i poslovnih informacija. Drugi važan aspekt je pouzdanost u radu, tj. raspoloživost i skalabilnost Veb servisa. Dva poznate platforme: J2EE i .NET možete koristiti da bi razvili aplikacije sa Veb servisima. To su veliki horizontalni radni okviri koji obezbeđuju, pored mnogih drugih stvari, zahtevanu funkcionalnost za razvoj interoperabilnih servisa. Obe platforme obezbeđuju fleksibilne, sigurne modele i druge mehanizme za podršku skalabilnosti i pouzdanosti.

Veb servisi nam omogućuju da zadovoljimo sledeće principe projektovanjanja softvera:

1. *Podeli i vladaj:* Aplikaciju čine neyavisni podasitemi koji su distribuisani i raspoloživi na Internetu.
2. *Povećati koheziju:* Veb servisi primenjuju slojevitu arhitekturu.
3. *Smanjiti međuzavisnost:* Veb aplikacije selabavo povezane.
4. *Povećati abstrakciju:* Klijenti servisa neznaju detaje oko načina implementacije veb servisa
5. *Povežati višestruku upotrebljivost komponenata.* Veb servisi su realizovani kao komponente ya višestruku upotrebu
6. *Povećaj ponovnu upotrebljivost:* Veb servisi su po prirodai ponovo upotrebljivi.
7. *Predvideti zastarelost:* Kada zastari postojeća tehnologija Veb servisa, može se primeniti nova. Međutim, korisnici to neće osetiti jer je način komunikacije ostao nepromenjen.
8. *Portabilnost:* Veb servisi se mogu rimeniti na raznim raunarim koji podržavaju Veb standarde.
9. *Lako testiranje:* Svaki servis ili usluga može se nezavisno testira.
10. *Defanzivno defanzivno:* Aplikacije pisane od različitih programera mogu da koriste isti servis.

ARHITEKTURE APLIKACIONIH SISTEMA

Aplikacioni sistemi su softverski sistemi koji se koriste radi zadovoljenja poslovnih ili organizacionih potreba neke firme.

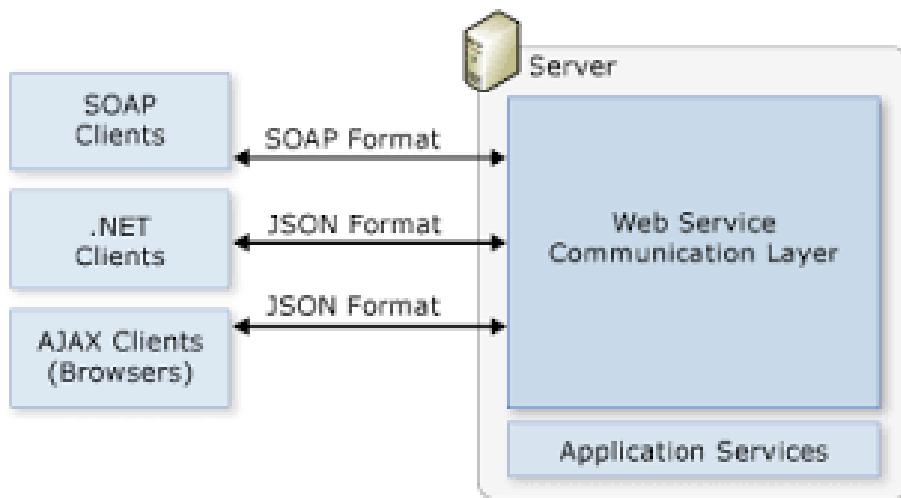
Aplikacioni sistemi su softverski sistemi koji se koriste radi zadovoljenja poslovnih ili organizacionih potreba neke firme. Firme u istom poslovnom sektoru imaju slične potrebe, te i aplikacioni sistemi u tim sektorima imaju sličnu funkcionalnost, tj. skup funkcija koje podržavaju. Ovi sistemi imaju softverske arhitekture koje odražavaju strukturu i organizaciju premereno organizacijama u koje obavljaju slične funkcije.

Aplikacione arhitekture sadrže glavne karakteristike određene klase softverskih sistema. Zato, postoje proizvodi koji imaju „standardnu“ arhitekturu za određenu klasu biznisa, a za potrebe svake organizacije u toj klasi, ti softverski sistemi imaju mogućnost prilagođavanja. Na taj način, firme ne moraju da razvijaju svoje, nove sisteme, već mogu da kupe odgovarajući standardni sistem i da ga onda prilagode svojim potrebama. Po pravilu, to bi trebalo da bude jeftinije, brže i pouzdano rešenja. Na primer, na ovaj način se najčešće danas primenjuju sistemi sa upravljanje poslovanjem organizacija, koji se nazivaju ERP sistemima (Enterprise Resource Planning). Najveći proizvođači tih sistema su danas SAP i Oracle.

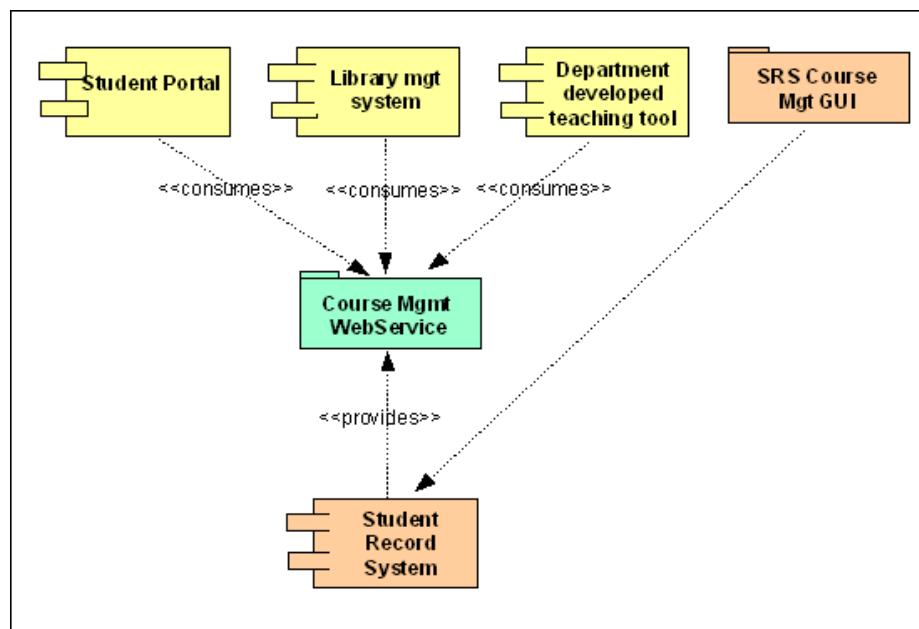
Na tržištu se nude gotovi paketi (softverski sistemi) za pojedine uobičajene poslovne funkcije, tj. specijalizovane aplikacije za određene specifične poslove. Ovi sistemi se u konkretnoj situaciji konfigurišu i prilagođavaju konkretnim potrebama korisnika sistema. U tome mogu učestvovati specijalizovani spoljni konsultanti, koji predlažu najbolji način primene određenog opštег softverskog sistema.

PRIMERI

Primeri servisno-orientisane arhitekture



Slika 4.6.2 SOA primer



Slika 4.6.3 SOA primer

ZADATAK

Primena arhitektonskih šablonu

1. Zašto se koriste nekoliko šablon za projektovanje arhitekture, u slučaju razvoja velikog softverskog sistema? Koje dodatne informacije, sem šablonu, mogu biti od koristi pri projektovanju velikih sistema?
2. Predložite arhitekturu sistema za prodaju muzike preko Interneta. Koje arhitektonski šabloni su osnova za njihovu arhitekturu?
3. Potražite na Internetu koja nudi javne Veb servise. Opišite servis koji nudi i dajte primer aplikacije koja bi mogla da ga koristi.

▼ Poglavlje 5

Arhitekture aplikacija

KORIŠĆENJE MODELAA ARHITEKTURA APLIKACIJA

Postoje više načina korišćenja modela arhitekture aplikacije.

Ako ste projektant softvera možete koristiti model arhitektura aplikacije na različite načine:

1. *Kao početnu tačku procesa projektovanja arhitekture:* Ako nedovoljno poznajete vrstu aplikacije koju treba da razvijete, možete uzeti, kao početni uzor, arhitekturu opšte aplikacije za tu vrstu poslovanja, i onda prilagođavate tu arhitekturu potrebama aplikacije koju razvijate.
2. *Kao referencu za upoređenje:* Pošto ste razvili arhitekturu sistema koji projektujete, želite da uporedite vaše rešenje sa arhitekturom opštih aplikacija u tom domenu.
3. *Kao način da organizujete rad projektnog tima:* Imajući u vidu standardnu arhitekturu aplikacija u određenom domenu, možete raspodeliti zadatke razvoja arhitektura vašeg sistema po uobičajenim komponentama koje će i vaša arhitektura imati.
4. *Kao sredstvo za ocenu komponenti radi višestruke upotrebljivosti:* Pri projektovanju komponente sistema, želite da je napravite tako da se može koristiti i u drugim sistemima, tj. da se višestruko koristi. Zato je korisno da se ona upoređuje sa komponentama opštih sistema da bi ste videli da li ona ima svojstva koja su slična svojstvima komponentama opštih sistema. Ako ima, to je dobar indikator da se i vaša komponenta može koristiti i u drugim aplikacijama.
5. *Kao rečnik termina koji se koristi kod razgovora o aplikacijama određenog tipa:* Kod diskusija o specifičnim aplikacijama ili kod upoređivanja aplikacija istog tipa, možete koristiti koncepte opštih arhitektura kada govorite o aplikacijama.

ZAJEDNIČKE ARHITEKTURE TIPOVA APLIKACIJA

Aplikacije istog mogu se opisati zajedničkom arhitekturom koju sve aplikacije određenog tipa imaju.

U praksi se koristi veliki broj različitih aplikacija (softverskih sistema razvijenih za određene poslove). Međutim, te aplikacije se u stvari ne razlikuju tako mnogo kao što na prvi pogled to izgleda. Imaju mnoge zajedničke elemente. Zato, one se mogu opisati zajedničkom arhitekturom koju sve aplikacije određenog tipa imaju. Radi ilustracije, ovde se navode dve takve arhitekture:

1. **Aplikacije za transakcione obrade:** To su aplikacije u kojima centralno mesto zauzima baza podataka. Aplikacija obrađuje zahtev za upis ili dobijanje informacije dobijenu od

korisnika , u skladu sa tim, vrši promene u bazi podataka, odnosno isporučuje traženu informaciju korisniku (posle određene obrade, tj. pripreme izveštaja). Ovi tipovi aplikacija su najčešće primenjeni kod interaktivnih poslovnih sistema. One su tako organizovane da akcija korisnika ne može da utiče na akcije drugih korisnika, a integritet baze podataka je uvek očuvan. Primeri takvi sistema su: interaktivni bankarski sistemi, sistemi e-poslovanja, informacioni sistemi organizacija, ili sistemi za rezervacije.

2. Sistemi za obradu jezika: U ovim sistemima korisnik se izražava primenom nekog formalnog jezika (npr u Javi). Sistem obrađuje ovaj jezik i pretvara ga u neki unutrašnju format (oblik) i onda interpretira (predstavlja) to kao svoje unutrašnje predstavljanje korisničkog iskaza. Na primer, tako rade tzv, kompjajleri, tj. prevodioci programskih jezika, koji prevode programe viših programskih jezika u unutrašnji mašinski kod računara. Pored toga, ovaj tip sistema može da prevodi komande namenjene sistemima baza podataka (npr. SQL) ili jezika za markiranja (npr. XML).

PRIMERI

Primeri zajedničkih arhitektura tipova aplikacija

Aplikacije za transakcione obrade:

- Primer je sistem za rezervaciju bioskopskih karata na internetu. Sistem funkcioniše sa jednom bazom podataka u kojoj nikakve izmene neće narušiti njen integritet i celokupna komunikacija se ostvaruje posredstvom te baze.
- Informacioni sistem banke

Sistemi za obradu jezika:

- Kompajleri programskih jezika

ZADATAK

Odgovorite na sledeća pitanja

1. Objasniti i navesti primer aplikacije za transakcione obrade.
2. Objasniti i navesti primer sistema za obradu jezika.

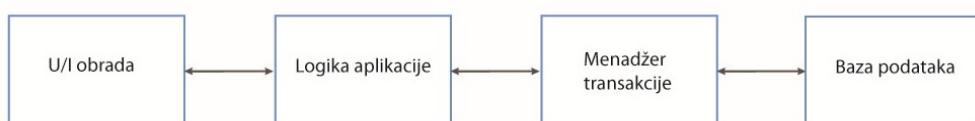
5.1 Sistemi za transakcionu obradu

TRANSAKCIJE I STRUKTURA TRANSAKCIIONIH SISTEMA

Sistemi za transakcionu se projektuju da obezbede obradu zahteva korisnika za informacijama iz baze podataka, ili zahteve za unos novih podataka u bazu podataka

Sistemi za transakcionu obradu (engl. *Transaction processing*- TP) se projektuju da obezbede obradu zahteva korisnika za informacijama iz baze podataka, ili zahteve za unos novih podataka u bazu podataka. Transakcija u sistemu baza podataka je niz operacija koji se tretira kao jedna jedinica (atomska jedinica) koja se mora izvršiti u celini pre nego što se stanje u bazi podataka trajno promeni (tj. upisani podaci postaju memorisani). To obezbeđuje integritet baze podataka, jer i u slučaju nekog poremećaja u toku procesa upisivanja novih podataka ili čitanja postojećih, ne može doći do nedefinisanih stanja baze. Transakcija (definisan niz operacija) mora se o celosti izvršiti, ili se poništavaju izvršene operacije niza i sistem se postavlja na početak, tj. u stanje u kome je bio pre početka izvršenje prve operacije u transakciji. Sistemi za transakcionu obradu su najčešće u formi interaktivnih sistema u kojima korisnici postavljaju asinhronne zahteve za određene servise sistema. .

Na slici 1 prikazana je koncepcija arhitektura jedne TP aplikacije. Najpre korisnika postavlja zahtev preko U/I komponente sistema. Taj zahtev se onda obrađuje primenom odgovarajuće logike aplikacije. Kreira se transakcija kojom rukovodi komponenta „Menadžer transakcija“. To je najčešće komponenta sistema za upravljanje bazama podataka (engl. DBMS - Database Management Systems). Kada menadžer transakcija utvrdi da je transakcija u celosti završena, on šalje signal aplikaciji da je obrada zahteva završena



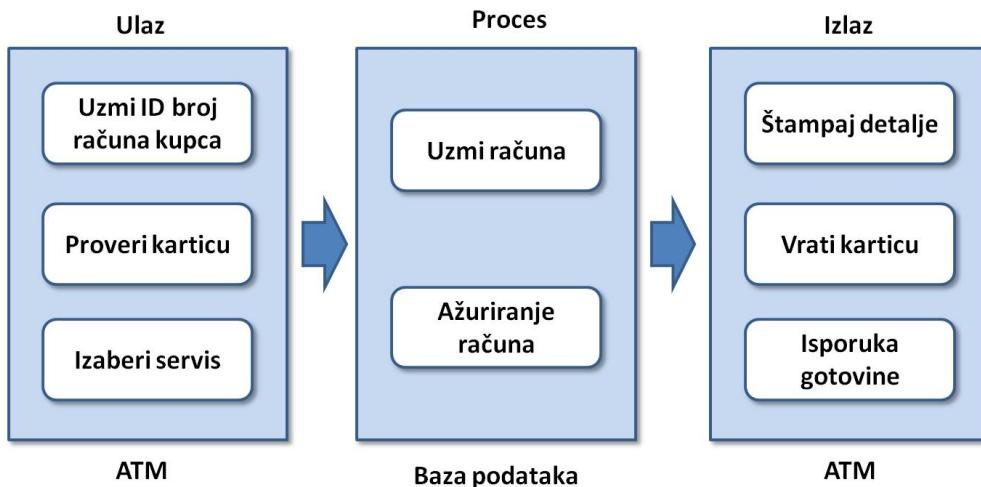
Slika 5.1.1 Struktura sistema za transakcionu obradu

PRIMER TRANSAKCIJONOG SISTEMA – ATM SISTEM

Transakcija postaje kompletirana i konačno izvršena tek kada su sve operacije predviđene za realizaciju neke funkcije urađene bez problema i prekida.

Sistemi za transakcionu obradu često se projektuju po šablonu arhitekture „cevi i filtera“, te sistem ima komponente koje su odgovorne za ulaz zahteva i podataka, obradu podataka i izlaz traženih rezultata. Jedan primer transakcionog sistema je softverski sistem bankomata (engl. ATM – *Automated Teller Machine*). Na slici 2 prikazana je arhitektura ATM sistema. Ulazne i izlazne komponente su primenjene softverski u okviru bankomata (ATM) a obradu podataka računa vrši bankarski sistema za upravljanje bazama podataka.

Transakcija postaje kompletirana i konačno izvršena tek kada su sve operacije predviđene za realizaciju neke funkcije urađene bez problema i prekida. Tek tada se promeni stanje na računu štediše, ako je povukao ili prebacio novac na neki drugi račun.



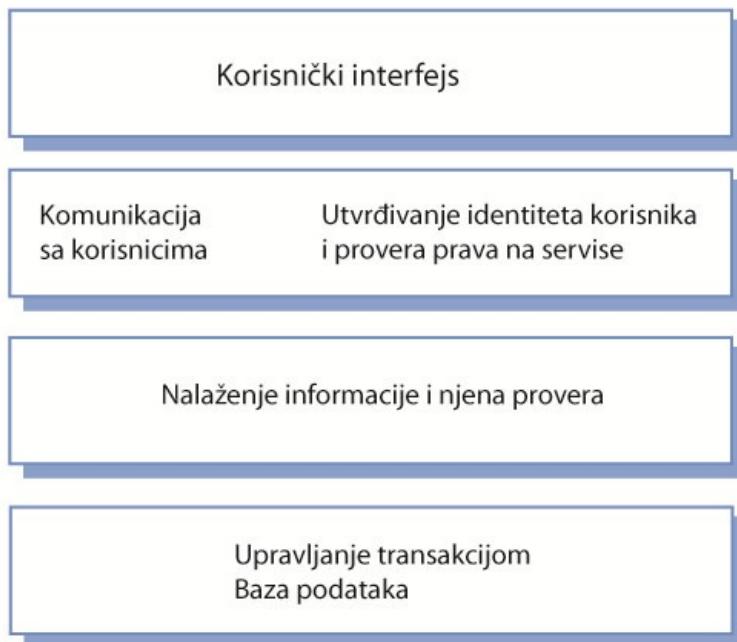
Slika 5.1.2 Arhitektura ATM softverskog sistema

ARHITEKTURA INFORMACIONIH SISTEMA

Svi sistemi koji koriste interakciju sa zajedničkom bazom podataka su tzv. transakcionalni informacioni sistemi.

Svi sistemi koji koriste interakciju sa zajedničkom bazom podataka su tzv. transakcionalni informacioni sistemi. Informacioni sistem dozvoljava kontrolisan pristup velikoj bazi informacija, kao što je katalog bibliotečkih jedinica, red vožnje letova, ili elektronski zdravstveni kartoni pacijenata u nekoj bolnici. Najčešće, informacioni sistemi su i veb sistemi, jer obezbeđuju pristup preko veb pretraživača.

Na slici 1 prikazan je jedan vrlo uopšteni model jednog informacionog sistema. Sistem koristi slojevitu arhitekturu, u kojoj sloj na vrhu obezbeđuje interfejs sa korisnicima, a najniži sloj, sa bazom podataka sistema. Komunikacioni sloj obezbeđuje sve izlaze i izlaze iz korisničkog interfejsa, a sloj za prikupljanje informacija koristi specifičnu logiku aplikacije za pristup bazi i za upisivanje novih podataka u njoj.



Slika 5.1.3 Slojevita arhitektura informacionog sistema

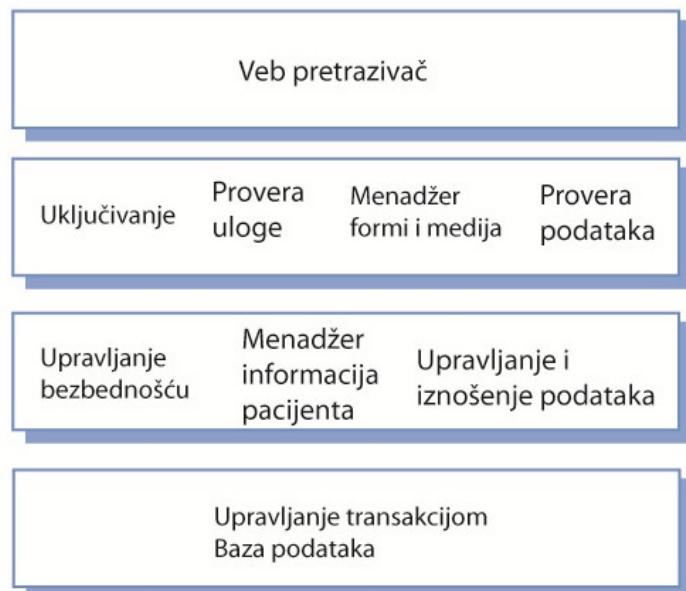
PRIMER INFORMACIONOG SISTEMA: MHC-PMS SISTEM

Sistem održava i upravlja informacijama o pacijentima sa mentalnim problemima

Jedan primer informacionog sistema (MHC-PMS) sa slojevitom arhitekturom prikazan je na slici 2. Sistem održava i upravlja informacijama o pacijentima sa mentalnim problemima:

1. Sloj na vrhu je odgovoran za korisnički interfejs, koji je preko veb pretraživača.
2. Drugi sloj obezbeđuje funkcionalnost korisničkog interfejsa. Sadrži komponente za uključenje korisnika i koje proveravaju da li korisnik koristi samo dozvoljene operacije, u skladu sa svojom ulogom. Ovaj sloj upravlja formama na kojima se predstavljaju informacije korisnicima, a komponenta za validaciju podataka proverava konzistentnost informacija.
3. Treći sloj primenjuje funkcionalnost sistema jer ima komponente koje obezbeđuju bezbednost sistema, kreiranje i održavanje informacija o pacijentima, a daje na izlazu podatke o pacijentima za prebacivanje u druge baze podataka, kao i komponente koje vrše generisanje izveštaja o radu sistema.
4. Najniži sloj, sadrži sistem za upravljanje bazom podataka (DBMS) koji obezbeđuje upravljanje transakcijama i trajno skladište podataka

Sloj koji obezbeđuje upravljanje podacima, može da koristi ne samo jednu, već više servera sa sistemima za upravljanje podacima, zavisno od potrebama aplikacije. To dovodi do efikasnog upravljanja velikim brojem transakcija i baza sa velikim brojem podataka.



Slika 5.1.4 Arhitektura MHC-PMS sistema

ZADATAK

Komponente informacionog sistema

Upotrebom osnovnog modela arhitekture informacionih sistema, datom na predavanju, predložite komponente koje bi mogle biti deo informacionog sistema koji bi dozvolio korisnicima da vide informacije o dolascima i odlascima aviona na određenom aerodromu.

Svoj odgovor pošaljite instruktoru.

▼ Poglavlje 6

Projektna dokumentacija

ZAŠTO DOKUMENTACIJA? KOME JE NAMENJENA DOKUMENTACIJA?

Dokumentacija dovodi do boljeg projektnog rešenja i poboljšava komunikaciju sa njegovim korisnicima.

Projektna dokumentacija ima dve svrhe:

1. Omogućava projektantnu ili projektnom timu da donosi dobre odluke u vezi projektnog rešenja, jer omogućava dodatno razmišljanje o onome što je projektovano.
2. Omogućava komunikaciju o projektnom rešenju sa drugima.

Pri izradi dokumentacije mogu se učiti slabosti u projektnom rešenju. Pri recenziji rešenja, takođe semože doći do poboljšanja. Inženjer softvera često izbegavaju izradu dokumentacije, ili je rade kada je projekat na kraju. To je velika greška, jer se kasnije, mogu otkriti manjkavosti u projektnom rešenju. Zamislite da neko gradi zgradu ili prozvodi automobil bez dokumentacije? U svim oblastima inženjerstva, tehnička dokumentacija je preduslova za uspešan rad i realizaciju projekta,. Tako treba da bude i u softverskom inženjerstvo. Kao mlada inženjerska disciplina, i ona mora da poštuje pravila i iskustva drugih inženjerskih disciplina.

Početi programiranje aplikacije, bez prethodno pripremljene dokumentacije, može dovesti do nefleksibilnosti sistema i do prevelike složenosti softverskog sistema.

Pri pisanju dokumentacije, morate znati kome se obraćate dokumentacijom. Projektnu dokumentaciju će koristiti sledeće tri ljudi:

- **Programeri**, koji treba da primene projektno rešenje sistema
- **Inženjeri softvera** koji će u budućnosti menjati sistem
- **Inženjeri softvera** koji razvijaju podsistem koje, preko interfejsa, treba da bude povezan sa vašim sistemom.

Zavisno od osoba kojima je namenjen dokument, ili deo dokumenta, projektan odlučuje koje informacije da uključi u dokumentaciju.

Važno je da se ne daje samo gotovo, urađeno projektno rešenje. Pored toga, važno je navesti tazloge zbog kojih je prepremljeno ovakvo rešenje. Čitalac će onda bolje shvatiti projektno rešenje. Recenzenti će lakše proveriti da li su donete dobre odluke projektanata, a inženjeri doržavanja će lakše moći da predlože usavršavanja projektnog rešenja.

SADRŽAJ PROJEKTNOG DOKUMENTA

Dokument se piše u skladu ciljnom grupom čitalaca, i treba da ima i odgovarajuće informacije a i odgovarajuću dužinu.

Preporučuje se da dokument sadrži sledeće informacije:

1. **Svrha:** Navedite koji sistem ili deo sistema opisuje dokument. Dajte reference ka zahtevima koji su primenjeni.
2. **Opšti prioriteti:** Navedite prioritete koje ste koristili pri izradi projektnog rešenja.
3. **Sažet opis projektnog rešenja:** Dajte sažeti opis projektnog rešenja da bi ga čitaoc brzo razumeo. Dijagrami u tome moogu biti od velike koristi.
4. **Glavna projektna pitanja:** Iznesite važna pitanja na koje ste nalazili odgovore. Navedite i moguće alternative koje ste razmatrali, i dajte razloge za odluke koje ste doneli.
5. **Detalji projektnog rešenja:** Navedite svi informacije koje čitalac treba da zna, a koje do sada niste dali. Ovde možete uključiti detaljne opise protokola za komunikaciju zmeđu klijenta i servera, opis strukture podataka i algoritama, kao i upotrebu različitih API.

Generalno, kada pišete projektni dokument, gledajte da ne bude ni isuviše kratak, ali ni isuviše dug. Primena dokumenta odgovarajuće dužine omogućava uspešno prenošene informacije onima kojima je dokument namenjen. Ne zaboravite da ne traga davati informacije koje čitaoc već zna te ih nikada neće čitati IYbegnite davanje informacija koje čitalac može lako dobiti iz drugih izvora.

:Obratite pažnju na sledeće preporuke:

- Izbegnite opis informacije koja je nepotrebna, jer je poznata, vešt tom programeru ili projektantu.
- Izbegnite da pišete detalje u projektnom dokumentu,koje je bolje pisati u vidu komentara u programskom kodu.
- Izbegnite da pišete o detaljima koji s emogu automatki dobiti iy samog programskega koda. kao što je lista javnih metoda.

Kako se kod često menja, zato je potrebno te komentare pisati u samom kodu, da ne bi se kasnije morale izmene vršiti na dva mesta: u samom kodu, ali i u projektnom dokumentu.

✓ Poglavlje 7

Zaključak

ZAKLJUČAK

Pouke u vezi arhitekture softvera

1. Arhitektura softvera je opis organizacije nekog softvera. Od njegove arhitekture zavise svojstva sistema, kao što su performanse, bezbednost i raspoloživost.
2. Odluke projektovanja arhitekture obuhvataju odluka o tipu aplikacije, distributivosti sistema, arhitektonskom stilu koji će se koristiti, i o načinima na kojima će se dokumentovati i vrednovati arhitektura
3. Arhitekture softvera se mogu dokumentovati koristeći nekoliko različitih pogleda na softver, kao što su: koncepcijski pogled, logički pogled, procesni pogled, razvojni pogled i fizički pogled.
4. Arhitektonski šabloni su sredstvo višestrukog korišćenja opštih arhitektura sistema. One opisuju arhitekturu, objašnjavaju kada mogu da budu upotrebljavane, i diskutuju prednosti i nedostatke koje nude.
5. Česti arhitektonski šabloni su: Model-pogled-kontroler, Slojevita arhitektura, Skladište, Klijent-server, i Cev i filter.
6. Opšti modeli arhitektura aplikacionih sistema pomažu razumevanje rada aplikacija, upoređenje sa drugim aplikacijama istog tipa, odobravanje projektnog rešenja informacionog sistema, i višestruku upotrebljivost komponenta.
7. Sistemi za obradu transakcija su interaktivni sistemi koji omogućuju udaljeni pristup informacijama u bazi podataka i njihove promene od strane svojih korisnika. Primeri transakcionih obradnih sistema su informacioni sistemi i sistemi za upravljanje resursima.
8. Sistemi za obradu jezika se upotrebljavaju za prevođenje teksta iz jednog jezika u drugi i za prevođenje instrukcija definisanih ulaznim jezikom. To može biti prevodilac i neka apstraktna mašina koja izvršava generisan jezik.