Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Nikolina Stupjak

Dizajniranje strukture sustava

Seminarski rad

Osijek, 2019

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Nikolina Stupjak

Dizajniranje strukture sustava

Seminarski rad

Kolegij: Softversko inženjerstvo

Voditelj kolegija: doc. dr. sc. Alfonzo Baumgartner

Osijek, 2019.

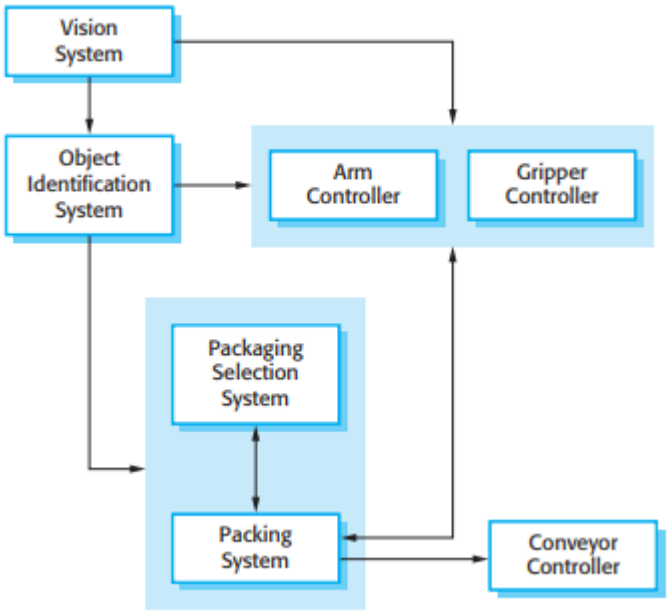
Sadržaj

Sažetak

U ovom seminarskom radu predstaviti ćemo kako dizajnirati strukturu sustava. Pokazat ćemo zašto je dizajn strukture sustava bitan i koje odluke trebamo donjeti o strukturi pri dizajniranju. Predstaviti ćemo neke često korištene modele te neke od načina za organiziranje strukture softvera koji se ponovno mogu koristiti pri dizajniranju novih sustava.

1 Uvod

Dizajniranje strukture softvera proces je koji se bavi proučavanjem kako bi sustav trebao biti organiziran i dizajniranjem cijele stukture sustava. To je prvi korak u dizajniranju softvera i ključna je poveznica između dizajna i potreba sustava, jer identificira glavnu strukturu komponenata u sustavu i veze među njima. Dizajniranje strukture sustava je proces koji kreira strukturni model koji opisuje kako će sustav biti organiziran kao skup komponenti koje međusobno komuniciraju.



Slika 1.1: Struktura sustava robota za pakiranje

Na slici je prikazana struktura sustava robota za pakiranje sa svim komponentama. Ovaj robot može podići nekoliko različitih objekata, koristi komponentu za vid kako bi podigao objekt ili identificirao kojeg je tipa taj objekt. Sustav onda prenosi objekt na pakiranje. Strukturni model sa slike prikazuje sve komponente sustava i relacije među njima.
U praksi, postoji preklapanje između obrade zahtjeva na sustav i dizajniranja strukture sustava. Idelano, zahtjevi na sustav nebi trebali sadržavati podatke o dizajnu, no to je moguće samo za jako male sustave. Preciznije, kao dio obrade zahtjeva može se zatražiti apstraktna struktura sustava, pri čemu onda možemo pridružiti funkcionalnosti komponentama sistema.
Strukturu softvera možemo dizajnirati na dva nivoa, koje ćemo zvati struktura u malom i struktura u velikom.

- Struktura u malom je struktura individualnih programa. Promatra se kako su oni podjeljeni na komponente.
- Struktura u velikom je struktura kompleksnih sustava koji koriste druge sustave, programe i programske komponente. Takvi veliki sustavi mogu se nalaziti na različitim računalima, kojima vlasnici mogu biti različite kompanije.

Struktura softvera je bitna, jer direktno utječe na preformanse, robusnost, distrbuciju i održivost samog sustava. Komponente ćemo kreirati tako da ispunjavaju funkcionalne zahtjeve na sustav. Ne funkcionalni zahtjevi ovise o samoj strukturi sustava, odnosno o načinu na koji su te komponente organizirane i kako komuniciraju. U mnogim sustavima, na nefunkcionalne zahtjeve utjecaj će također imati i same kompnente, ali najveći utjecaj ima sama struktura sustava. Tri su prednosti dizajniranja i dokumentiranja strukture softvera:

- *Komunikacija među sudionicima* Struktura je high-level prezentacija sustava koja može biti korištena kao glavni kriterij za donošenje odluka među sudionicima.
- *Analiza sustava* Izrada strukture sustava u ranoj fazi izrade sustava zahtjeva detaljnu analizu. Odluke dizajniranja strukture sustava imaju utjecaj na to hoće li sustav zadovoljiti zahtjeve koji se tiču preformansi, pouzdanosti i održivosti.
- *Ponovno korištenje* Model strukture sustava je kompaktan izvršiv opis kako bi sustav trebao izgledati i kako su komponente međusobno povezane. Struktura sustava je često ista za sustave koji imaju slične ili iste zahtjeve, i stoga se može ponovno koristiti. Može se dogoditi da je ista struktura korištena na velikom broju povezanih sustava.

Dizajniranje strukture softvera u početku može služiti kao plan za dizajniranje sustava, biti pomoć pri pregovaranju o zahtjevima na sustav, a kasnije se može koristiti i u odlukama o samom strukturiranju u sustavu. Dizajn strukture softvera nije pisan u detalje, ali dizajnerima daje do znanja koje su ključne komponente u sustavu. Strukturu sustava najčešće modeliramo koristeći blok dijagrame, kao u primjeru koji smo ranije pokazali. Blok unutar bloka sugerira da je komponenta podjeljena na podkomponente. Strelice označavaju da se podatci prosljeđuju među komponentama u smjeru strelice. Blok dijagrami se koriste upravo zato što su jednostavni za razmjevanje svim osobama koje su uključene u razvoj sustava. Mana ovakvih reprezentacija je što ne pruikazuju detaljno cijelu arhitekturu, ne prikazuju tipove veza između komponenata niti sama svojstva komponenti.
Do kontradikcije između teorije u stukturiranju sustava i prakse dolazi zbog sljedeća dva načina na koji se model strukture programa koristi:

- *Kao oblik razmatranja dizajna sustava* High-level strukturalni prikaz sustava je koristan za komunikaciju sa sudionicima i planiranje projekta, jer takav prikaz nije pretrpan detaljima. Svi sudionici na projektu mogu razumjeti ovakav apstraktni prikaz sustava, bez da su zbunjeni s nepotrebnim detaljima. Modeli sturkture sustava identificiraju ključne komponente koje trebaju biti izrađene kako bi te komponente bile dodjeljene timovima na izradu.
- *Kao oblik dokumentiranja onoga što je dizajnirano* Cilj je izraditi cijeli model sustava koji prikazuje različite komponente sustava, njihova sučelja, i veze među njima. Prednost ovog načina je da ovakav detaljan opis strukture pomaže pri razumjevanju i izradi sustava.

Blok dijagrami su prikladan način za opisivanje strukture sustava pri dizajniranju, obzirom da su oni prikladni za komunikaciju među sudionicima u izradi sustava. Na većini projekata to je zapravo jedina dokumentacija o strukturi koja postoji. Međutim, ukoliko se struktura sustava temeljito dokumentira tada je bolje koristiti dobro definiranu terminologiju, ali smatra se da detaljna dokumentacija nije niti poželjna niti korisna.

2 Odluke u dizajniranju strukture sustava

Dizajniranje strukture sustava je proces u kojem dizajniramo i organiziramo sustav kako bi on zadovoljio funkcionalne i nefunkcionalne zahtjeve koje imamo. Aktivnosti unutar procesa ovise o tipu sustava kojeg izrađujemo, prethodnom iskustvu arhitekta sustava, i zahtjevima koje imamo. Dizajniranje strukture sustava je stoga bolje smatrati nizom odluka nego nizom aktivnosti. Tokom dizajniranja strukture sustava, arhitekti sustava trebaju donjeti odluke o strukturi koje utječu na sustav i njegovu izradu. Obzirom na njihovo prethodno iskustvo, oni trebaju u obzir uzeti i sljedeće stvari:

- Postoji li gotova struktura koju možemo koristit kao predložak za sustav koji izrađujemo?
- Kako će sustav biti distribuiran?
- Koje modele strukture sustava i stilove možemo koristiti?
- Kakav pristup koristiti u strukturiranju sustava?
- Kako će komponente u sustavu biti podjeljene na podkomponente?
- Kako ćemo kontrolirati operacije unutar komponenata?
- Kakva je organizacija sustava najbolja kako bi ispunili nefunkcionalne zahtjeve?
- Kako ćemo ocjeniti je li dizajn strukture sustava dobar?
- Kako ćemo dokumentirati strukturu sustava?

Bez obzira što je svaki softver jedinstven, sustavi koji se koriste za slične stvari često imaju slične arhitekture. Prilikom dizajniranja strukture sustava, trebamo razmisliti što naš sustav i sustavi koje smo prije izrađivali imaju zajedničko i što od njih možemo koristiti.
Struktura sustava može bit bazirana na određenom modelu ili stilu. Model strukture sustava je opis organizacije sustava i on sadrži bitne informacije o strukturi koja je korištena u drugom sustavu. Pri odabiru načina strukturiranja biramo onakav stil koji će zadovoljniti zahtjeve na sustav koje imamo. Kako bi sustav rastavili na dijelove, trebamo pronaći način kako podjeliti komponente na podkomponente Na kraju, kontroliramo modeliranje sustava, i odlučujemo kako ćemo kontrolirati izradu samih komponenata.
Prilkom odabira modela strukture sustava trebamo voditi računa da struktura direktno utječe na nefunkcionalne zahtjeve na sustav i to na sljedeći način:

- *Preformanse* Ukoliko su preformanse ključan zahtjev, tada struktura treba biti dizajnirana tako da se ključne operacije odvijaju unutar malog broja komponenti, pri čemu će se te komponente nalaziti se po mogućnosti unutar jednog računala, odnosno neće biti raspoređene unutar cijele mreže. To može značiti da trebamo koristiti relativno malo velikih komponenti umjesto više malih, što smanjuje broj komunikacija među komponentama.
- *Zaštita* Ukoliko je zaštita ključan zahjev, treba se koristiti slojevita struktura, pri čemu se podatci koji trebaju biti zaštićeni nalaze u najdubljim slojevima, s visokim stupnjem zaštite na tim slojevima.

- *Sigurnost* Ukoliko je sigurnost ključan zahtjev, tada struktura treba biti dizajnirana tako da su sve operacije koje su orjentirane na sigurnost unutar jedne komponente ili unutar malog broja komponenti. To smanjuje troškove i probleme sa ispitivanjem sigurnosti.Poželjno je postoji i dodatna zaštita koja gasi cijeli sustav ukoliko je ugrožena sigurnost.

- *Dostupnost* Ukoliko je dostupnost ključan zahtjev, tada struktura sustava treba biti dizajnirana tako da uključuje ponavljajuće komponente,što omogućuje zamjenu i ažuriranje komponenata bez zaustavljanja sustava.

- *Održivost* Ukoliko je održivost ključan zahtjev tada struktura treba biti dizajnirana tako da se koriste male komponente, koje se lako mogu mjenjati. Podatci koji se koriste u izradi trebaju biti odvojeni od korisničkih podataka te se trebaju izbjegavati zajednički podatci .

Naranvno, postoje potencijalni konflikti među nekima od ovih struktura. Na primjer, korištenje velikih komponenata poboljšava preformanse a korištenje malih komponenata poboljšava održivost. Ukoliko su i preformanse i održivost bitni zahtjevi na sustav, mora se nepraviti kompromis. To se može postići tako da se koriste različiti modeli strukture sustava za različite dijelove sustava. Provjera kvalitete dizajna strukture sustava je težak posao, jer pravi test strukture je koliko sustav ispunjava tražene funkcionalne i nefunkcionalne zahtjeve kada se sustav koristi. Međutim, testiranje se može izvršiti na način da trenutni dizajn uspoređujemo s prje korištenim dizajnima.

3 Pristupi dizajniranju strukture sustava

Modeli softverskog sustava mogu se koristiti kako bi se donjele odluke o zahtjevima na sustav ili o dizajnu. Alternativno, mogu se koristiti kako bi dokumentirali dizajn kako bi on mogao biti ponovno korišten ili kako bi ga dalje nadogradili u novom sustavu. U ovom poglavlju, dotaknut ćemo se dva problema koja se mogu pojaviti u oba slučaja:

- Kakav je model koristan pri dizajniranju i dokumentiranju strukture sustava?

- Kakvu notaciju trebamo koristiti za opisivanje modela strukture sustava?

Nemoguće je predstaviti sve bitne informacije o strukturi sustava u jednom modelu, obzirom da svaki model prikazuje samo jedno stajalište ili jedan pristup dizajnu sustava.Model može pokazivati kako je sustav podjeljen po modulima, kako procesi međusobno komuniciraju ili neke druge načine kako su komponente distribuirane kroz mrežu. Sve od navedenog je korisno ali u raličitim trenutcima, tako da i dizajn i dokumentaciju obično trebamo korz nekoliko različitik pristupa. Postavlja se pitanje koje sve pristupe treba uzeti u obzir. Postoje 4 osnovna pristupa, a to su:

- *Logički pristup* Pokazuje nam ključne objekte ili klase objekata u sustavu. Mora postojati mogućnost da se zahtjevi na sustav povežu s entitetima u ovom pristupu.

- *Proceduralni pristup* Pokazuje nam kako je sustav sastavljen od procesa koji međusobno surađuju. Ovaj pristup je koristan za donošenje odluka o nefunkcionalnim karakteristikama sustava kao što su preformanse i dostupnost.

- *Razvojni pristup* Pokazuje nam kako je sustav rastavljen za razvoj, tj. kako je sustav podjeljen na komponente koje će razvijati jedan developer ili njegov tim.

- *Fizički pristup* Pokazuje nam kako su hardverske i softverske komponente podjeljene po procesorima u sustavu.

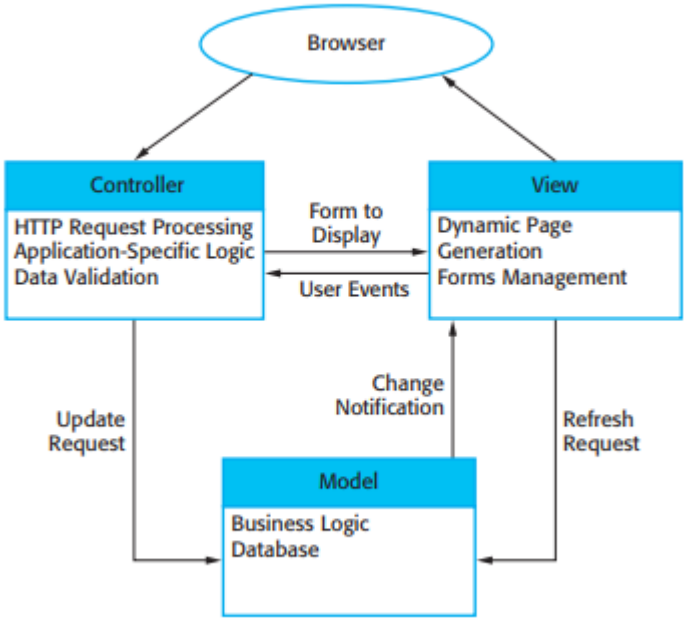
Uz ova 4 pristupa, smatra se da bi korisno bilo uzeti još jedan, tzv. konceptualni pristup. Ovakv pristup je apstraktan pogled na sustav koji može biti baza za dekompoziciju high-level zahtjeva na detaljnije specifikacije. Može pomoći inženjerima u donošenju odluka koje komponente se mogu ponovno koristiti. Primjer iz uvoda (1.1) je primjer konceptualnog pristupa dizajnu sustava. U praksi, konceptualni pristup se gotovo uvijek koristi tokom dizajniranja i koristi se kako bi pomogao donušenju oduka o strukturi sustava. Također, to je dobar način kako predočiti bit sustava svim sudionicima. Tijekom procesa dizajniranja, mogu biti korišteni i ostali pristupi za neke dijelove sustava, ali nema potrebe za izradom detaljnih opisa svih pristupa.
Također, postavlja se pitanje trebaju li softverski arhitekti koristiti UML za opis stukture sustava. Pokazalo se da u praksi, da kada se UML koristi, koristi se većinom samo u neformalnim oblicima. Do toga dolazi i zbog činjenice da je UML dzajniran za opisivanje objektno orjentiranih sustava, i da tokom dizajniranja strukture sustaca želimo jako dobro opisati sustav, ali i dalje na apstraktnoj raznini. Klase objekata su usko vezane s implementacijom i zbog toga ih je teško koristiti za opis arhitekture. Preporuka je da se u ovom slučaju koriste specijalizirani arhitektalni jezici (**ADL**) kako bi opisali arhitekturu sustava. Osnovni elementi ADL-a su komponente i poveznice, i uključuju pravila i upute za dobro kreiranu arhitekturu. Međutim, obzirom da su takvi jezici specijalizirani samo za jedno područje, nisu razmljivi velikom broju ljudi, te je teško iskoristiti njihov potpuni potencijal u softverskom inženjerstvu. Upravo iz tog razloga, smatra se da će UML i dalje ostati najčešće korišteni način za dokumentiranje strukture sustava.
Agilne metode razvoja sustava tvrde nema potrebe za detaljnom dokumentacijom dizajna jer ona se poslije neće koristiti, te da je to gubitak novca i vremena za njenu izradu. Za većinu sustava, ova tvrdnja je točna, i nije potrebno izrađivati detaljan opis strukture za sva četiri navedena pristupa. Prilikom izrade dokumentacije o strukturi sustava, trebamo brinuti o izradi dokumentacije koju ćemo koristiti u komunikaciji, ali se ne moramo brinuti je li ta dokumentacija potpuna. Međutim, iznimku treba napraviti kada radimo ključne sustave, i tada trebamo napraviti detaljnu analizu sustava i zahtjevati da dokumentacija strukture sustava bude potpuna.

4 Modeli strukture sustava

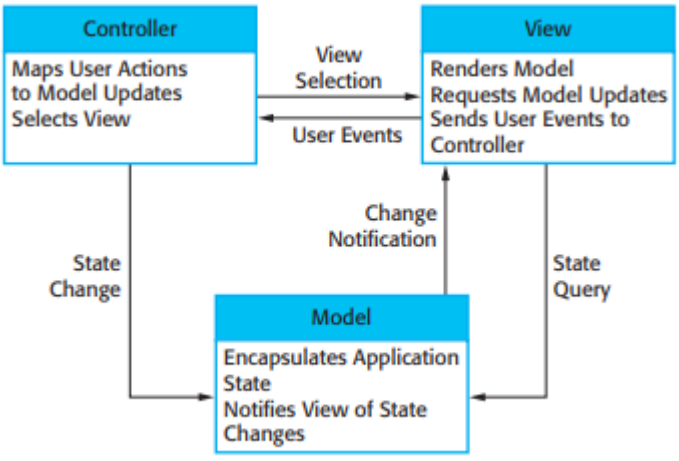
Modeli su oblik prezentacije, dijeljenja i ponovnog korištenja znanja o sustavu, no nisu često korišteni. U ovom poglavlju, predstaviti ćemo što su to modeli i ukratko opisati neke od njih koji se koriste u različitim tipovima sustava.Modele strukture sustava možemo smatrati stiliziranim apstraktnim opisom dobre odluke, koja je isprobana i testirana u različitim sustavima i okolinama. Stoga, amodel strukture sustava treba opisivati organizaciju sustava koja se pokazala dobrom u prijašnjim sustavima. Treba sadržavati inoformacije o tome kada je i kada nije dobro koristiti takav model i koje su jake i slabe strane tog modela.

MVC (Model-View-Controller)	
Opis	Odvaja vizualizaciju i interakciju od podataka u sustavu. Sustav je strukturiran u tri logičke komponente koje međusobno komuniciraju. Komponenta Model upravlja podatcima u sustavu i njima pridruženim operacijama. Komponenta View definira i upravlja kako su podatci predstvaljeni korisniku. Komponenta Controller upravlja korisničkom interakcijom i prosljeđuje te akcije komponentama View i Model
Primjer	Slika4.2 predstavlja strukturu WEB aplikacije uz korištenje MVC uzorka
Korištenje	Koristi se kada postoji više načina kako prikazati i upravljati s podatcima. Također koristi se kada ne znamo koji su zahtjevi za upravljanje i prikaz podataka
Prednosti	Omogućava da se podatci mjenjaju neovisno o prikazu i suprotno. Podržava prikaz podataka na različite načine kada se dogodi neka promjena u jednom od prikaza, te se onda odražava na sve prikaze
Mane	Može zahtjevati dodatnu implementaciju i onda kada to nije potrebno

Slika 4.1: MVC model



Slika 4.2: Arhitehtura web aplikacije uz korištenje MVC modela



Slika 4.3: Organizacija MVC

Slika 4.1 prikazuje Model-View-Controller model. Ovaj model je osnova za kontroliranje interakcije u mnogim WEB sustavima. Ovaj silizirani model sadrži ime modela, kratak opis s pridruženim grafičkim modelom i primjer gdje je ovaj uzorak već korišten. Također, uključeno je kada bi bilo dobro koristiti ovaj model te koje su njegove prednosti i mane. Grafički modeli koji su povezani s MVC modelom prikazani su na slikama 4.2 i 4.3. Oni prikazuju ovaj model iz različitih pristupa dizajniranju. Slika 4.3 prikazuje konceptualni pristup, dok je na slici 4.2 run-time struktura kada je uzorak korišten za interakciju u web sustavu.

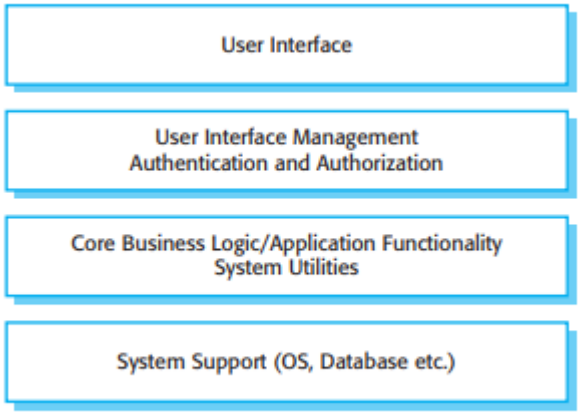
4.1 Slojeviti model

Pojmovi podjele i nezavisnosti su ključni u dizajniranju strukture sustava jer dozvoljavaju da sve promjene budu lokalizirane. MVC uzorak, prikazan na slici 4.1 dijeli sustav na elemente kako bi dozvolio da je promjene događaju neovisno. Na primjer, dodavanje novog prikaza ili mjenjanje postojećeg može se izvršiti bez promjena u podatcima unutar modela. Slojeviti model stukture sustava je još jedan od načina kako to možemo postići.

Slojevita struktura	
Opis	Organizira sustav po slojevima s povezanim funkcionalnostima priduženim svakom sloju. Svaki sloj prosljeđuje servise na sloj iznad tako da najniži slijevi predstavljaju ključne servise koje će se koristiti kroz sustav.
Primjer	Slojeviti model sustava za dijeljenje autorskih prava među različitim knjižnicama, slika 4.6
Korištenje	Koristi se kada se dodaje novi sadržaj na već postojeći sustav; kada je razvoj podjeljen među nekoliko timova od kojih svatko ima odgovornost za jedan sloj funkcionalnosti; kada postoje zahtjevi za višeslojnom sigurnošću.
Prednosti	Omogućava zamjenu cijelog sloja dokle god je sučelje nepromjenjeno. Ponavljajući sadržaj može se proslijediti svakom sloju kako bi se povećala nezavisnost sustava.
Mane	U praksi, teško je pronaći način kako podjeliti sustav na slojeve, i najviši sloj trebao bi komunicirati direktno s najnižim a ne kroz slojeve. Preformanse mogu biti smanjene zbog više slojeva interpretacije i zahtjeva servisa koji se izvršavaju na svakom sloju.

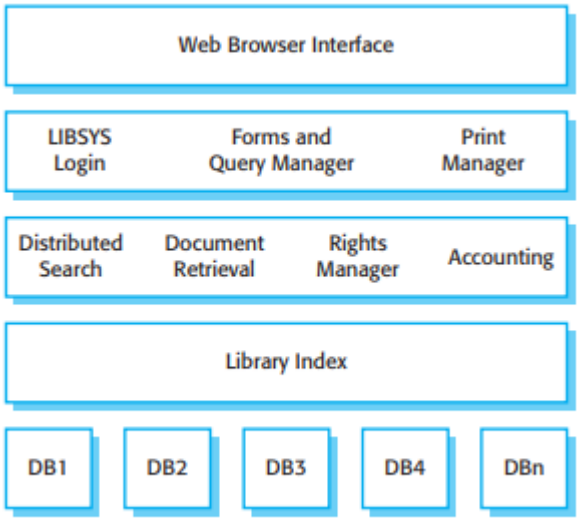
Slika 4.4: Slojeviti model

Na slici 4.4 je prikazan model slojevite strukture sustava. Funkcionalnost sustava je organizirana kroz različite slojeve, gdje svaki sloj ovisi o sadržaju i servisima koji su mu proslijeđeni iz sloja direktno ispod njega. Ovakav slojeviti pristup omogućuje inkrementalni razvoj sustava. Kada je jedan sloj razvijen servisi razvijeni kasnije mogu biti dostupni korisnicima. Struktura je promjenjiva i prenosiva sve dok je sučelje nepromjenjeno, tj. jedan sloj može biti zamjenjem drugim ekvivalentnim slojem. Nadalje, ukoliko se mjenja sloj ili su novi slojevi dodani na vrh, promjene se dešavaju samo u susjednim slojevima. Na slici 4.5 je primjer slojevite strukture s 4 sloja. Najniži sloj najčešće sadrži



Slika 4.5: Generički slojeviti model

bazu podataka ili operacijski sustav. Sljedeći sloj je aplikacijski sloj koji sadrži komponente koje su povezane s funkcionalnošću aplikacije i korisne komponente koje koriste druge aplikacijske komponente. Treći sloj sadrži komponente koje su povezane s korisničkim sučeljem i podržava korisničku autorizaciju i autentifikaciju. Zadnji sloj rezerviran je za korisničko sučelje. Naravno, broj slojeva je proizvoljan. Svaki od ova četiri sloja mogao je biti podjeljen u dva ili više slojeva.



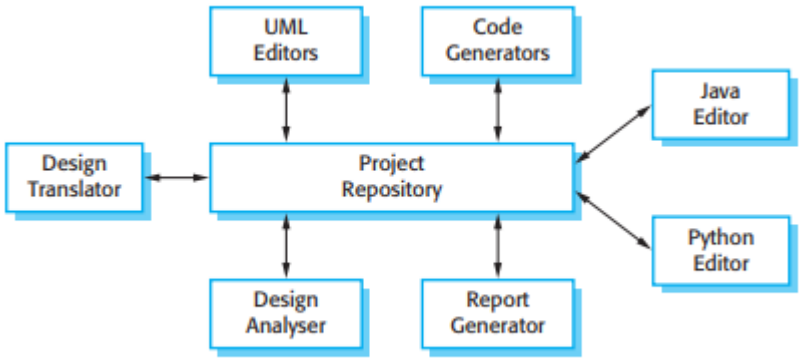
Slika 4.6: Struktura LIBSYS sustava

Slika 4.6 primjer je kako ovakav slojeviti model strukture sustava se može upotrijebiti u sustavu za knjižnice zvanom LIBSYS, koji omogućava elektronički pristup materijalima iz više sveučilišnih knjižnica. On ima peteroslojnu arhitekturu, u kojoj se u najdubljem sloju nalaze baze podataka u svakoj knjižnici.

4.2 Model s repozitorijem

Model s repozitorijem	
Opis	Svim podacima sustava se upravlja iz centralnog repozitorija iz kojeg se može pristupiti svim komponentama sustava. Komponente međusobno direktno ne komuniciraju, ali komuniciraju kroz repozitorij.
Primjer	Slika 4.8 predstavlja primjer IDE-a gdje komponente koriste repozitorij za podatke o dizajnu sustava. Svaki softverski tool generira podatke koji su dalje dostupni za korištenje drugim tool-ovima.
Korištenje	Ovaj uzorak bi se trebao koristiti kada imamo sustav u kojem se generira jako velika količina podataka.
Prednosti	Komponente mogu biti neovisne, a promjena jedne komponente može se preko repozitorija odraziti na ostalim komponentama ukoliko je to potrebno.
Mane	Cijeli sustav ovisi o repozitoriju, stoga problemi u repozitoriju se odražavaju na cijeli sustav. Također može biti neefikasno organizirati cijelu komunikaciju kroz repozitorij.

Slika 4.7: Model s repozitorijem



Slika 4.8: Model s repozitorijem

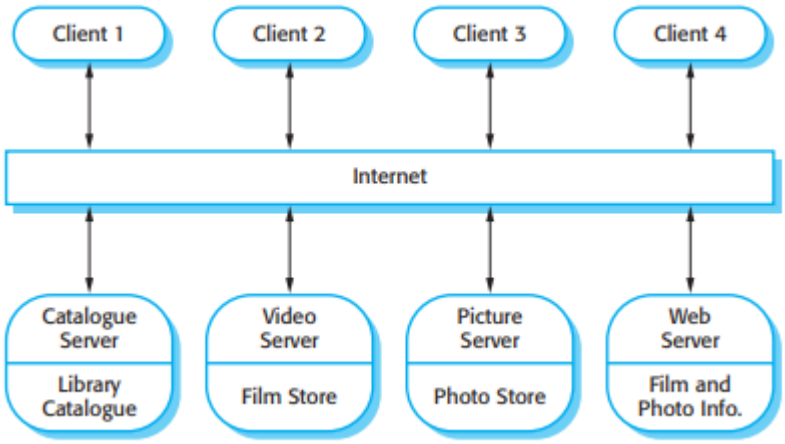
Veliki broj sustava s velikom količinom podataka organiziran je oko jedne dijeljene baze podataka ili repozitorija. Ovakav model je stoga prikladan za korištenje kada su podatci generirani od strane jedne komponente a ostale komponente ih koriste. Slika 4.8 je ilustracija situacije u kojoj se može koristiti repozitorij. Dijagram prikazuje IDE koji sadrži različite toolove. Repozitorij u ovom slučaju prati sve promjene na softveru i dozvoljava vraćanje na prethodne verzije. Ovakav model, tj. ako su toolovi organizirani oko repozitorija je učinkovit način za djeljenje velike količine podataka jer nema potrebe za direktnim prebacivanjem podataka iz jedne komponente u drugu. Međutim, sve komponente komuniciraju preko repozitorija. U ovakvom modelu najveći problem je što je teško ili nemoguće dodavati nove komponente ukoliko modeli njihovih podataka ne odgovaraju danoj shemi. Na istom primjeru, možemo primjetiti da je repozitorij pasivan i da kontrolu nad njim vrše ostale komponente. Alternativni pristrup, koji je razvijen za AI sustavee, koristi model 'ploče' koji obavještava komponente kada su pojedini podatci dostupni. Takav pristup je dobro koristiti kada je model s repozitorijem loše strukturiran.

4.3 Model Klijent-Poslužitelj

Mana modela s repozitorijem što je repozitorij statičan. Zbog toga uvodimo sljedeći model:

Model klijent-poslužitelj	
Opis	U Klijent-Poslužitelj modelu, funkcionalnost sustava je organizirana po servisima pri čemu svaki servis dolazi od drugog servera.
Primjer	Slika 4.10 predstavlja primjer videoteke organizirane kao klijent-poslužitelj sustav.
Korištenje	Koristi se kada se dijeljenoj bazi podataka treba pristupati s više lokacija.
Prednosti	Glavna prednost ovakvog modela je da poslužitelji mogu biti raširenи u mreži. Funkcionalnosti mogu biti dostupne svim korisnicima i ne trebaju biti razvijene na svim serverima.
Mane	Preformanse mogu biti nepredvidljive jer ovise o mreži koliko i o sustavu. Također može doći do problema u menadžmentu ukoliko su poslužitelji u vlasništvu različitih tvrtki.

Slika 4.9: Model klijent-poslužitelj



Slika 4.10: Model klijent-poslužitelj za videoteku

Sustav koji je rađen po modelu klijent-poslužitelj je organiziran kao skup servisa i pridruženih poslužitelja te klijenata koji mogu pristupiti i koristiti servise. Najveće komponente ovakvog modela su:

- Skup poslužitelja koji pružaju servise ostalim komponentama.
- Skup klijenata koji pozivaju servise koje poslužitelji pružaju.
- Mreža koja dozvoljava da klijenti pristupe poslužiteljima.

Važna prednost ovakvog modela su podjeljenost i neovisnost. Servisi i poslužitelji se mogu mjenjati bez utjecaja na ostale dijelove sustava. Klijenti bi trebali znati imena dostupnih poslužitelja i koje servise oni pružaju. Međutim, poslužitelji ne moraju identificirati klijente ili znati koliko klijenata pristupa njihovim servisima. U suštini, klijent proslijeđuje zahtjev na poslužitelja i čeka dok ne dobije odgovor. Slika 4.10 je primjer sustava koji se temelji na takvom modelu. Ovo je sustav s više korisnika, baziran na WEBu koji se koristi za pristup filmovima ili fotografijama. U ovom sustavu, nekoliko servera upravlja i prikazuje različite tipove podataka. Video mora biti emitiran brzo i sinkronizirano ali u relativno niskoj rezoluciji. Mogu biti kompresirani u videoteci, tako da poslužitelj koji obrađuje video može ga dekompresirati u različitim formatima. S druge strane slike moraju ostati u visokoj rezoluciji, te je prikladno obrađivati ih na drugom poslužitelju. Najeća prednost ovakvog modela je da je njegova struktura dijeljena. Lako je dodati novog poslužitelja u već postojeći sustav ili nadograditi poslužitelje bez da se utječe na ostale dijelove sustava.

5 Zaključak

Dizajniranje strukture sustava je proces koji povezuje dizajn i potrebe sustava. Prilikom dizajniranja strukture sustava važno je uzeti u obzir sve nefunkcionalne zahtjeve na sustav. Obzirom na zahtjeve koje imamo odabrat ćemo i pristup koji ćemo koristiti u dizajniranju strukture. Za slične sustave korisno je koristiti slične ili čak iste strukture. Pri tome možemo koristiti neke od poznatih modela, a koji od njih ćemo odabrati ovisi najviše o zahtjevima koje trebamo ispuniti.

Literatura

[1] I. Sommerville, *Software Engineering (6th edition)*, Addison Wesley Publ. Co., USA, 2000.

[2] *https://codeburst.io/software-architecture-the-difference-between-architecture-and-design-7936abdd5830*

[3] *https://en.m.wikipedia.org/wiki/Software_architecture*

[4] Krutchen, P. (1995). *The 4+1 view model of software architecture*