# DIZAJN ARHITEKTURE

## ARHITEKTURA SOFTVERA

**Dizajn arhitekture** je razvojni proces u kojem se identificiraju:

* pod-sustavi promatranog sustava
* okosnice za njihovu kontrolu i međusobnu komunikaciju.

Rezultat tog procesa je **opis arhitekture softvera**.

Jedna od početnih faza procesa dizajna sustava. Predstavlja vezu između procesa dizajna i specifikacije.

* Često se izvodi zajedno s nekim aktivnostima specifikacije.

Za vrijeme dizajna se izrađuju brojni modeli arhitekture, pri čemu svaki prikazuje sustav iz različite perspektive.

## PREDNOSTI JASNE DEFINIRANE ARHITEKTURE

* Komunikacija među zainteresiranim stranama
* Arhitektura se koristi kao apstrakcija višeg nivoa za komunikaciju između različitih zainteresiranih strana.
* Analiza sustava
* Izrada jasne arhitekture u ranoj fazi projekta zahtjeva određene analize iz čega će biti moguće zaključiti hoće li sustav zadovoljiti kritične zahtjeve kao performanse, pouzdanost ili jednostavnost održavanja.
* Ponovno korištenje komponenti
* Opis jedne ili više komponenti sustava i njihove međusobne komunikacije se često može iskoristiti za sustave sa sličnim zahtjevima te je moguće ponovno korištenje istih komponenti.

## UTJECAJ ARHITEKTURE NA KARAKTERISTIKE SUSTAVA

Arhitektura sustava utječe na njegove performanse, robusnost, jednostavnost distribucije i održavanje.

Ne-funkcionalni zahtjevi mogu odrediti stil i strukturu koji će se odabrati za modeliranje nekog sustava:

* **Performanse**

Lokalizirati kritične operacije i minimizirati komunikaciju (koristiti manje velikih komponenti).

* **Zaštita**

Koristiti slojevitu arhitekturu, pri čemu su kritične komponente u unutarnjim slojevima

* **Sigurnost**

Dijelovi sustava na koje se primjenjuju ista sigurnosna ograničenja se trebaju nalaziti unutar istog podsustava ili unutar malog broja podsustava

* **Dostupnost**

Uključivanje redundantnih komponenti i drugih mehanizama kako bi sustav bio što otporniji na greške

* **Održavanje**

Koristiti više malih, zamjenjivih komponenti.

## SUKOBI INTERESA KOD ODABIRA ARHITEKTURE

* Korištenje manje velikih komponenti poboljšava performanse, ali otežava održavanje.
* Uvođenje redundantnih podataka poboljšava dostupnost ali je teže osigurati njihovu sigurnost.
* Lokaliziranje dijelova sustava s istim sigurnosnim ograničenjima znači više komunikacije što utječe na performanse.

## STRUKTURIRANJE SUSTAVA

* Bavi se dekompozicijom sustava u niz pod-sustava koji su u međusobnoj interakciji.
* Svaki podsustav može biti sustav sam za sebe.
* Struktura sustava se najčešće prikazuje kao blok dijagram koji daje pregled arhitekture sustava.
* Uz to može prikazivati na koji način podsustavi izmjenjuju podatke, kako su distribuirani i izgled njihovih sučelja.

## BLOK DIJAGRAM

* Vrlo apstraktan prikaz, ne prikazuje na koji su način komponente povezane, niti karakteristike podsustava.



* Baš zbog toga je koristan za planiranje projekta

(ukazuje na ključne elemente) i komunikaciju

sa zainteresiranim stranama.

## ODLUKE KOJE JE POTREBNO DONIJETI KOD DIZAJNA ARHITEKTURE

1. Postoji li kakva općenita arhitektura sustava koja se može koristiti?
2. Kako će sustav biti distribuiran?
3. Koji stil arhitekture treba koristiti?
4. Na koji način će se sustav razbiti na module?
5. Koja kontrolna strategija će se koristiti?
6. Kako će se procijeniti kvaliteta dizajna arhitekture?
7. Na koji način će arhitektura biti dokumentirana?

## MODELI ARHITEKTURE

* Koriste se za dokumentiranje dizajna arhitekture.
* Koriste se:
* **Statički modeli** – prikazuju glavne komponente sustava – strukturu sustava
* **Dinamički modeli** – prikaz procesa koji se odvijaju za vrijeme rada sustava
* **Modeli sučelja** – opisuju sučelja pod-sustava
* **Modeli veza** – dijagrami toka podataka koji prikazuju veze između podsustava
* **Modeli distribucije** – prikazuju kako su podsustavi distribuirani po fizičkim komponentama (računalima).

## STILOVI ARHITEKTURE

* Odražavaju osnovnu strategiju koja se koristi za strukturiranje sustava.
* Tri osnovna, najčešće korištena stila su:
* Model repozitorija sa zajedničkim podacima
* Klijent – server model
* Slojeviti model

## MODEL REPOZITORIJA

* Koristi se kod aplikacija u kojima podsustavi dijele velike količine podataka.
* Zajednički podaci se nalaze u centralnoj bazi podataka ili repozitoriju kojem mogu pristupiti svi podsustavi.
* Pasivna struktura u kojoj je kontrola nad podacima dana podsustavima koji koriste taj repozitorij.



## PRIMJER – ARHITEKTURA CASE ALATA

Prednosti:

* Efikasan način dijeljena velike količine podataka jer nema potrebe za prosljeđivanjem podataka iz jednog sustava u drugi.
* Podsustavi se ne moraju brinuti o kako podatke koriste ostali podsustavi.
* Upravljanje podacima je centralizirano (stvaranje sigurnosnih kopija, sigurnost, …).
* Model dijeljenja podataka se objavljuje preko sheme repozitorija.

Nedostatci:

* Svi podsustavi se moraju dogovoriti o modelu podataka u repozitoriju (uvijek je kompromis).
* Stvaranje novih podataka ili izmjene postojećih je jako složeno i skupo.
* Različiti podsustavi mogu imati različite zahtjeve npr. za sigurnošću, dok model repozitorija postavlja istu politiku za sve.
* Javlja se problem efikasne distribucije sheme repozitorija zbog redundancije i nekonzistencije.

## KLIJENT – SERVER MODEL

Model koji se koristi za distribuirane sustave a prikazuje raspodjelu podataka i modula za njihovu obradu na čitavom nizu komponenti.

Sastoji se od:

* Jednog ili više servera koji pružaju određene usluge (ispis, upravljanje podacima, …)
* Jednog ili više klijenata koji koriste te usluge
* Mreže koja omogućava pristup klijenta serveru.

Klijenti moraju znati imena dostupnih servera i njihovih usluga, dok serveri ne moraju imati nikakve informacije o klijentima.



## Primjer – Biblioteka filmova i fotografija

## KARAKTERISTIKE KLIJENT – SERVER SUSTAVA

Prednosti

* Jasna raspodjela podataka
* Moguće korištenje jeftinijeg hardvera zbog efikasne upotrebe mreže
* Jednostavno dodavanje novih servera ili nadogradnja postojećih

Nedostatci

* Ne postoji zajednički model podataka pa podsustavi koriste različitu organizaciju podataka što može otežati razmjenu podataka
* Redundantno upravljanje svakim serverom
* Ne postoji centralni registar imena servera i usluga koje pružaju, pa može biti teško pronaći koji su sve serveri i usluge dostupne.

## SLOJEVITI MODEL



* Naziva se još i eng. Abstract machine model
* Koristi se za modeliranje sučelja podsustava jer organizira

sustav u niz slojeva (nivoa) od kojih svaki pruža

određeni niz usluga.

Primjer: APSE sustav za upravljanje verzijama

## KARAKTERISTIKE SLOJEVITOG MODELA

Prednosti:

* Podržava inkrementalni razvoj sustava, na način da kada je gotov neki sloj usluge koje on pruža mogu biti dostupne klijentima.
* Jednostavna zamjena slojeva – jedino moramo sačuvati sučelja prema susjednim slojevima.
* Ukoliko se promjeni sučelje nekog sloja to utječe samo na susjedne slojeve.

Nedostatci:

* Često je apstraktno sustav modelirati na ovaj način.
* Niže slojeve često ne koriste direktno samo susjedni slojevi nego i ostali viši slojevi.
* Poruka se od nekog višeg sloja mora provući kroz više slojeva koji su nepotrebni – pa to utječe na performanse.

## MODULARNA DEKOMPOZICIJA

Postupak pri kojem se radi dekompozicija podsustava u module.

* **Podsustav** – komponenta osnovnog sustava koju možemo promatrati kao samostalnu cjelinu i čiji rad ne ovisi o radu drugih podsustava.
* **Modul** – komponenta sustava koja pruža neke usluge ali se ne može promatrati kao odvojeni sustav.

Dva osnovna modela modularne dekompozicije:

* **Objektni model** – sustav se prikazuje kao niz objekata koji međusobno izmjenjuju poruke.
* **Model toka podataka ili cjevovoda** – sustav se dijeli na niz funkcionalnih modula koji ulazne podatke transformiraju u izlazne.

## OBJEKTNI MODELI



Objektno-orijentirana dekompozicija koja se

bavi identifikacijom klasa objekata, njihovih

atributa i operacija.

Primjer – Sustav za obradu računa

## KARAKTERISTIKE OBJEKTNOG MODELA

Prednosti:

* Objekti su međusobno slabo povezani pa se njihova implementacija može mijenjati bez utjecaja na druge objekte.
* Objekti mogu predstavljati entitete iz stvarnog svijeta i mogu se ponovno koristiti.
* Objektno orijentirani jezici su široko rasprostranjeni.

Nedostatci:

* Promjene na sučelju jednog objekta uzrokuje probleme.
* Može biti problema s prikazom složenih entiteta.

## FUNKCIJSKI-ORIJENTIRANI CJEVOVODI

* Prikazuju se preko niza funkcionalnih transformacija koje obrađuju ulazne podatke kako bi dobili željeni izlaz.
* Naziva se još model cjevovoda (kao kod UNIX-a)
* Koriste se brojne varijante ovog modela kod kojih se transformacije mogu izvoditi sekvencijalno ili paralelno.
* Nije prikladan za interaktivne sustave.



**Primjer – Sustav za obradu računa**

## KARAKTERISTIKE MODELA CJEVOVODA

Prednosti:

* Podržava ponovno korištenje transformacija.
* Intuitivna organizacija za komunikaciju sa zainteresiranim stranama.
* Jednostavno dodavanje novih transformacija.
* Jednostavna implementacija kod istovremenih i slijednih sustava.

Nedostatci:

* Zahtijeva zajednički format podataka koji se kreću kroz cjevovod.
* Nije prikladan za prikaz interakcija baziranih na događajima.

## MODELI KONTROLE

-Opisuju tok kontrole između pod-sustava.

-Dva osnovna pristupa:

* **Centralizirana kontrola** – jedan pod-sustav je odgovoran za kontrolu rada i on pokreće i zaustavlja ostale pod-sustave.
* **Kontrola bazirana na događajima** – svaki pod-sustav može reagirati na vanjski generirane događaje bilo da su oni rezultat drugih pod-sustava ili okoline sustava.

## CENTRALIZIRANA KONTROLA

-Jedan kontrolni pod-sustav preuzima kontrolu za upravljanje izvršavanjem drugih pod-sustava.

-Ovisno tome radi li se o sustavu čiji pod-sustavi rade slijedno ili paralelno može se izabrati jedan od slijedećih modela:

* **Zovi – vrati** (eng. call – return) **model** – Model kod kojeg kontrola kreće od vrha programa i pomiče se prema dnu a pri tome kontrolu preuzima onaj potprogram koji je trenutno aktivan.
* **Model upravitelja** – jedna komponenta sustava kontrolira pokretane, zaustavljanje i koordinaciju ostalih komponenata.

**Zovi – vrati model Model upravitelja – kontrola sustava koji se izvode u stvarnom vremenu**



## SUSTAVI BAZIRANI NA DOGAĐAJIMA

* Sustavi koji su upravljani vanjskim događajima a pri tome pod-sustav koji odgovara na događaj nema nikakav utjecaj kada će se on desiti.
* Dva osnovna modela su:
* Model emitiranja (eng. broadcast model) – događaj se šalje svim podsustavima i oni podsustavi koji obrađuju te događaje mogu to i napraviti.
* Model upravljan prekidima (eng. interrupt-driven) – Koriste se u sustavima za radu u realnom vremenu gdje sustav za upravljanje prekidima detektira svaki prekid i šalje ga određenom podsustavu na obradu.

## MODEL EMITIRANJA

* Efikasan za integraciju pod-sustava na različitim računalima na mreži.



* Koristi sustav za upravljanje porukama i događajima.
* Svaki pod-sustav registrira kod sustava za upravljanje porukama i događajima kakvi ga događaji interesiraju i kada se on dogodi kontrola se prebacuje na pod-sustav koji može obraditi određeni događaj.
* Razlikuje se od centraliziranih sustava po tome što kontrola obrade nekog događaja nije dio sustava za upravljanje događajima i porukama.

## SUSTAVI UPRAVLJANI PREKIDIMA

* Koriste se u sustavima za rad u realnom vremenu gdje je važan vrlo brz odgovor na neki događaj.
* Postoje poznate vrste prekida koji se mogu desiti u definirano je upravljanje svaki od njih.
* Svaka vrsta prekida je povezana s određenom memorijskom lokacijom i hardverski preklopnik prebacuje kontrolu toka na komponentu zaduženu za upravljanje tim prekidom.
* Vrlo brz odziv ali je jako složeno za programiranje i validaciju.



**Sustavi upravljani prekidima**

## REFERENTNE ARHITEKTURE

* Neki modeli arhitekture mogu biti specifični za domenu određene aplikacije.
* Postoje dvije osnovne vrste takvih modela:
* **Generički modeli** – apstrakcija brojnih primjera stvarnih sustava a uključuju osnovne karakteristike tih sustava, mogu se direktno koristiti za dizajn;
* **Referentni modeli** – još apstraktniji, idealizirani modeli sustava koji pružaju informacije o toj vrsti sustava i služe za usporedbu različitih arhitektura.

****

**OSI referentni model**

# Arhitektura distribuiranih sustava

## Vrste sustava

* **Osobni sustavi** – nisu distribuirani, namijenjeni su za rad na osobnim računalima ili radnim stanicama.
* **Ugradbeni sustavi** (eng. embedded) – sustavi pokrenuti na jednom ili grupi više integriranih procesora a koriste se isključivo za jednu namjenu.
* **Distribuirani sustavi** – sistemski softver je pokrenut na grupi povezanih procesora/računala povezanih mrežom.
* To su sustavi kod kojih je obrada informacija distribuirana na nekoliko računala.
* Danas su svi veliki sustavi distribuirani.

## Prednosti distribuiranih sustava

* **Raspodjela resursa -** Dijeljenje hardverskih i softverskih resursa.
* **Otvorenost -** Korištenje opreme i softvera od različitih dobavljača.
* **Istodobnost** (eng. concurrency) - Istodobna obrada s ciljem popravljanja performansi .
* **Skalabilnost -** Povećanje protoka podataka dodavanjem novih resursa.
* **Otpornost na pogreške -** Sposobnost sustava da nastavi s radom jednom kada se dogodi pogreška.

## Nedostatci distribuiranih sustava

* **Kompleksnost -** Uglavnom kompleksniji od centraliziranih sustava, teže je predvidjeti izranjajuća svojstva i testirati ih.
* **Sigurnost -** Više su izloženi vanjskim napadima.
* **Upravljanje -** Potrebno je uložiti više truda za upravljanje takvim sustavima.
* **Nepredvidivost -** Nepredvidivi odziv ovisno o organizaciji sustava i mrežnom opterećenju.

## Arhitekture distribuiranih sustava

Osnovni tipovi arhitekture distribuiranih sustava su:

* **Višeprocesorske arhitekture** - Sustav se sastoji od više procesa koji se mogu (ali ne moraju) izvršavati na različitim procesorima.
* **Klijent-server arhitektura** - Sustav se sastoji od servera i klijenata pri čemu serveri pružaju niz servisa koje koriste klijenti.
* **Arhitektura distribuiranih objekata**

Ne postoji razlika između klijenta i servera, svaki objekt može

## Višeprocesorske arhitekture

* Najjednostavniji model distribuiranog sustava.
* Koristi se za prikaz arhitekture velikih sustava za rad u realnom vremenu.
* Raspodjela procesa po procesorima može biti unaprijed određena ili može biti pod kontrolom dispečera.



**Primjer – Višeprocesorski sustav za kontrolu prometa**

## Klijent – server arhitektura

* Model aplikacije se sastoji od niza servisa koje pružaju serveri i niza klijenata koji koriste te servise.
* Klijenti moraju znati koji serveri nude koje servise, dok server nema ne mora imati nikakve informacije o klijentima.
* Klijenti i serveri su logički procesi i njihova raspodjela po fizičkim komponentama ne mora biti 1:1.

**Logička arhitektura klijenata i servera Fizička arhitektura klijenata i servera**

## Slojeviti prikaz klijent – server arhitekture



Logička struktura klijent – server sustava koji se razvija se može podijeliti u tri sloja:

* **Prezentacijski** – sučelje prema korisnicima a koristi se za unos korisničkih

podataka i prezentaciju rezultata.

* **Aplikacijski** – implementira logiku aplikacije, ono što aplikacija radi.
* **Upravljanje podacima** – upravlja bazama podataka.

## Dvoslojna klijent – server arhitektura

Najjednostavniji prikaz klijent – server arhitekture u kojem se tri logička sloja smještaju na dvije komponente (klijent ili server).Ovisno o rasporedu razlikuje se:

* **Model mršavog klijenta** (eng. thin-client) – Server je zadužen za pokretanje aplikacije i upravljanje podacima, dok se klijent samo brine za komunikaciju s korisnikom.
* **Model debelog klijenta** (eng. fat-client) – Server jedino upravlja podacima, dok je na klijentu pokrenuta aplikacija i zadužen je za interakciju s korisnikom.

**Dvoslojna klijent – server arhitektura**

## Model mršavog klijenta

Koristi se:

* Jedno od rješenja kad centralizirani naslijeđeni sustav evoluira na klijent – server arhitekturu.

Naslijeđeni sustav je server zadužen za upravljanje podacima i izvođenje aplikacije a na klijentu se samo implementira grafičko sučelje.

* Koristi se kada su klijenti jednostavni mrežni uređaji (pokreće se samo preglednik), a ne računala.
* Glavni nedostatak je da poprilično opterećuje i mrežu i server (odgovoran za proračune i upravljanje podacima), a današnja računala na strani klijenata su uglavnom neiskorištena.

## Model debelog klijenta

* Server je isključivo zadužen za transakcije u bazi podataka.
* Klijent lokalno izvršava aplikaciju i brine se za interakciju s korisnikom.
* Prikladan za klijent- server sustave kod kojih se unaprijed zna kakve će biti karakteristike klijenata.
* Mnogo složeniji za upravljanje, jer se nove verzije aplikacije moraju instalirati na svim klijentima.



**Primjer – klijent - server sustav bankomata**



## Troslojna arhitektura

* Problem kod dvoslojne arhitekture je što se tri logička sloja pokušavaju

smjestiti na dvije fizičke komponente (klijent i server).

* Kod troslojne arhitekture svaki od logičkih slojeva se izvršava na svojoj
* fizičkoj komponenti.
* Osigurava bolje performanse od mršavog klijenta, a jednostavnija je
* za upravljanje od debelog.
* Skalabilna arhitektura kod koje se jednostavno dodaju



novi serveri.

**Primjer – Internet bankarski sustav**

****

**Primjeri korištenja klijent – server arhitekture**

|  |  |
| --- | --- |
| **Arhitektura** | **Primjena** |
| **Dvoslojna – mršavi klijent** | Naslijeđeni sustavi gdje nije praktično odvajanje aplikacije od upravljanja podacima.  Proračunski zahtjevne aplikacije (npr. kompajleri) kod kojih ima jako malo (nema) obrade poataka.  Aplikacije bazirane na podacima (pretraživanje, postavljanje upita) kod kojih nema nikakve obrade podataka. |
| **Dvoslojna –  debeli klijent** | Aplikacije kod kojih je obrada zasnovana na COTS sustavima na klijentu.  Aplikacije koje su zasnovane na proračunski zahtjevnoj obradi podataka.  Aplikacije sa stabilnim zahtjevima koje se koriste u sustavima s dobro organiziranim upravljanjem. |
| **Troslojna** | Aplikacije sa stotinama ili tisućama klijenata.  Aplikacije kod kojih se podaci i aplikacija često mijenjaju.  Aplikacije kod kojih se integriraju podaci iz više izvora. |

## Arhitektura distribuiranih objekata

* Ne postoji razlika između klijenta i servera.
* Svaki distribuirani entitet je objekt koji pruža servise drugim objektima i prima servise od njih.
* Komunikacija objekata  
  se odvija kroz među-sloj  
  (eng. mddleware) tzv.   
  posrednik za zahtjeve   
  objekata (eng. object   
  request broker - ORB).



## Arhitektura distribuiranih objekata

Arhitektura distribuiranih objekata je složenija za dizajn od klijent – server sustava.

Prednosti:

* Omogućava dizajnerima sustava da kasnije donesu odluku koji objekti i na kojim mjestima će pružati servise.
* Vrlo otvorena arhitektura koja dozvoljava jednostavno dodavanje novih resursa kada je to potrebno.
* Sustav je fleksibilan i skalabilan.
* Moguće je konfigurirati sustav dinamički na način da objekti migriraju preko mreže ukoliko je to potrebno.

## Gdje se koristi arh. dist. objekata?

* Logički model za strukturiranje i organizaciju sustava, pri čemu se misli kako ostvariti traženu funkcionalnost isključivo korištenjem servisa i kombinacija servisa.
* Fleksibilan pristup implementacije klijent – server sustava, pri čemu je logički model sustava klijent –server ali i jedan i drugi su realizirani kao distribuirani objekti koji komuniciraju preko zajedničkog komunikacijskog okvira.

## CORBA

* CORBA – **C**ommon **O**bject **R**equest **B**roker **A**rchitecture - međunarodni standard za ORB međusloj koji upravlja komunikacijom distribuiranih objekata.
* Kod distribuiranih računalnih sustava međusloj je potreban na dva nivoa:
* **Na nivou logičke komunikacije** međusloj dozvoljava objektima na različitim računalima izmjenu podataka i kontrolnih informacija
* **Na nivou komponenti** međusloj pruža osnovu za razvoj kompatibilnih komponenti – zbog čega je definiran CORBA standard za komponente.

## Struktura CORBA aplikacije

* **Aplikacijski objekti -**  dizajnirani i implementirani za određenu aplikaciju.
* **Standardni objekti** – definirao ih je OMG za određenu domenu (npr. financije/osiguranje, kupovinu preko mreže, …)



* **Osnovni CORBA servisi** – osiguravaju osnovne

servise neophodne za rad u distribuiranoj okolini

(npr. upravljanje direktorijima i sigurnošću)

* **Vodoravni CORBA sadržaji** – znači da su ovi sadržaji

zajednički različitim aplikacijskim domenama i koriste

se u različitim aplikacijama (npr. korisnička sučelja,

upravljanje sustavom, …)

## CORBA objekti

* CORBA objekti su vrlo slični objektima u C++, C#, Javi.
* Svaki objekt mora imati odvojenu definiciju sučelja napisanu korištenjem zajedničkog jezika (eng. Interface Definition Language - IDL).
* Postoji preslikavanje iz IDL u neke programske jezike kao što su C++, Java, …
* Objekti napisani u različitim programskim jezicima mogu međusobno komunicirati.

## ORB - Posrednik za zahtjeve objekata

* Upravlja komunikacijom objekata a da bi to bilo moguće mora

poznavati sve objekte u sustavu i njihova sučelja.

* Korištenjem ORB-a Objekt koji želi koristiti neki servis vezuje IDL

"stub" koji definira sučelje objekta koji nudi tu uslugu.

* Pozivanje ovog "stub-a" rezultira pozivom ORB-u,
* koji zatim poziva traženi objekt preko objavljenog
* IDL kostura koji povezuje sučelje i implementaciju servisa.

## Među-organizacijski računalni sustavi

Zbog sigurnosnih razloga većina distribuiranih računalnih sustava je organizirana na nivou poduzeća.

* Moguće je primijeniti lokalne standarde, menadžment i operacijske procese.
* Noviji modeli računalnih sustava su dizajnirani da podržavaju među-organizacijske računalne sustave gdje su različiti čvorovi locirani u različitim organizacijama.
* Dva karakteristična modela ovog pristupa su:
* P2P - ravnopravni (eng. peer-to-peer) računalni sustavi
* Servisno orijentirani sustavi

## Arhitektura P2P sustava

* P2P - ravnopravni (eng. peer-to-peer) sustavi su decentralizirani sustavi kod kojih se obrada može obaviti na bilo kojem čvoru u mreži, pri čemu ne postoji razlika između klijenta i servera.
* Sustav je dizajniran tako da se iskoristi prednost računalne snage i skladišnog prostora velikog broja umreženih računala.
* Najpoznatije primjene:
* Isporuka sadržaja
* Komunikacijske mreže (Skype, VoIP, ….)

## Modeli P2P arhitekture

* U osnovi u P2P sustavu bi svaki čvor trebao biti svjestan svih ostalih čvorova i biti u stanju uspostaviti vezu i izmjenjivati podatke s njim.
* U praksi to tako ne funkcionira nego se čvorovi organiziraju u lokalitete, s tim da onda neki čvorovi služe kao veza prema drugim lokalitetima
* Logička mrežna arhitektura P2P aplikacije se može podijeliti u dvije grupe:
* Decentralizirana
* Polu-centralizirana

## Decentralizirana P2P arhitektura

* Čvorovi su i funkcionalni elementi i služe za preusmjeravanje podataka i kontrolnih signala prema drugim čvorovima.
* Čvorovi su grupirani u lokalitete.
* Kada neko traži dokument, on se najprije traži unutar lokaliteta, ukoliko ga nema dokument se traži van lokaliteta i kada se pronađe dokument se usmjerava do čvora koji ga je tražio.



Ukoliko čvor n1 traži dokument koji se nalazi na n10, njegova potraga se preko čvora n3 usmjerava preko čvora n6, n9 do n10.

Decentralizirani sustav za upravljanje dokumentima

* Vrlo redundantna arhitektura, a time i otporna na greške tj. odspajanje čvorova iz mreže.

## Polu centralizirana P2P arhitektura

* Server pomaže uspostavu veze između čvorova u mreži
* Čvorovi na mreži uspostavljaju vezu sa serverom (iscrtane linije) da bi vidjeli koji su ostali čvorovi dostupni i nakon toga direktno s tim čvorovima uspostavljaju komunikaciju.



Sustav za slanje poruka u realnom vremenu   
(eng. instant messaging system)

## Servisno orijentirane arhitekture

* Zasnivaju se na korištenju već gotovih, dostupnih servisa (web servisi).
* Korištenje web servisa je standardan pristup izradi višestruko korištenih komponenti koje se dostupne preko web-a.
* Servis za prijavu poreza može pružiti korisnicima podršku za ispunjavanje obrasca za prijavu poreza i slanje poreznoj upravi.

## Web servisi



* Postoje razni modeli servisa (Jini, web i grid servisi, …)

ali svi oni su zasnovani na prikazanom modelu: 🡪

* Pružatelj servisa definira sučelje i implementira

funkcionalnost servisa, a zatim uključuje osnovne info.

o servisu u registru.

* Korisnik pronalazi željeni servis u registru, spaja se na

pružatelja i vezuje željeni servis u svoju aplikaciju.

## Standardi za izradu servisa



Servisi se baziraju na dogovorenom XML baziranom standardu kako bi

radili neovisno o platformi i programskom jeziku.

Osnovni su:

* SOAP – koristi se za komunikaciju među servisima.
* WSDL – standard za definiranje sučelja.
* WS-BPEL – koristi se za definiciju programskih

procesa koji uključuju više servisa.

## Servisi i distribuirani objekti

* Neovisnost pružatelja servisa.
* Dostupnost servisa se javno objavljuje .
* Potencijalno moguće vezivanje servisa za vrijeme izvršavanja.
* Stvaranje novih servisa korištenjem postojećih.
* Plaćanje servisa samo kada se koristi.
* Manje, kompaktnije aplikacije.
* Aplikacije koje brzo reagiraju i prilagođavaju se promjenama.

## Primjer – Scenarij servisa informacijskog sustava u automobilu

* Informacijski sustav u automobilu bi vozaču trebao pružiti informacije o vremenu, stanju na cesti i razne lokalne informacije.
* Povezan je na radio automobila pa se sve informacije

isporučuju na određenom kanalu.

* Automobil je opremljen GPS uređajem, pa na osnovu njegove pozicije pristupa se nizu informacijskih servisa.
* Vozač može odabrati jezik dostave obavijesti.

# Arhitektura aplikacije

## Generička arhitektura

Najčešće sustavi istog tipa imaju sličnu arhitekturu, a razlika između ovih sustava je samo u detaljnoj funkcionalnosti koju pružaju.

Korištenje generičke arhitekture:

* Početna točka za proces dizajna arhitekture
* Kao popis koje podsustave treba razviti
* Za organizaciju rada razvojnog tima
* Način za procjenu koje se mogu li se već gotove komponente iskoristiti ponovo.

## Tipovi aplikacija

* **Aplikacije za obradu podataka**
* Aplikacije koje pokreću podaci, tj. ove aplikacije obrađuju velike količine podataka u skupinama (eng. batch) bez eksplicitne intervencije korisnika.
* Sustav naplate, obrada plača, …
* **Aplikacije za obradu transakcija**
* Aplikacije razvijene oko BP, obrađuju korisničke zahtjeve i osvježavaju podatke u bazi.
* Sustav rezervacija, prodaja preko Interneta, …
* **Sustavi za obradu događaja**
* Aplikacije kod kojih reakcija sustava ovisi o interpretaciji događaja u okolini sustava
* Aplikacija za obradu teksta (Word), sustavi za rad u realnom vremenu (igre), …
* **Sustavi za obradu jezika**
* Aplikacije kod kojih se namjere korisnika specificiraju formalnim jezikom, kojeg zatim sustav interpretira i obrađuje.
* Prevodioci, interpreteri naredbi, …

## Aplikacije za obradu podataka

* eng. data-processing systems / batch-processing systems.
* Aplikacije organizirane oko obrade podataka pri čemu se koristi baza podataka znatno veća od samog softvera.
* Ulazni / izlazni podaci su organizirani u skupine (eng. batch) prikupljene iz datoteke ili baze podataka.
* Primjer sustav za slanje računa za električnu energiju:
* Ulaz – niz brojeva korisnika i očitanja brojila
* Izlaz – Odgovarajući niz računa, jedan za svakog korisnika
* Ovi sustavi najčešće imaju ulaz-obrada-izlaz strukturu.

## Ulaz-obrada-izlaz model

* **Ulazna komponenta** čita podatke iz datoteke ili baze podataka, provjerava njihovu ispravnost, ispravne slaže u red za obradu.
* **Komponenta za obradu** uzima transakcije iz ulaznog reda, obavlja potrebne proračune i stvara novi zapis s rezultatima proračuna.
* **Izlazna komponenta** čita zapise s rezultatima, formatira ih prema potrebi i upisuje u bazu podataka ili na pisač.



## Prikaz sustava za obradu podataka

Osnovna karakteristika sustava za obradu podataka je da se zapisi ili transakcije obrađuju serijski.

Pri tome ne postoji potreba da se sačuva stanje podataka dok se kreču kroz pojedine transakcije što znači da su ovakvi sustavi više funkcijski nego objektno orijentirani.

Najbolji način prikaza funkcijskih transformacija je korištenja dijagrama toka:

* Prikazuju na koji način se obrađuju podaci prolaskom kroz sustav.
* Transformacije se prikazuju elipsama, tok podataka – strelicama, pravokutnici – datoteke (ili neki drugi način pohrane podataka).

## Primjer: Dijagram toka sustava za isplatu plaća



## Aplikacije za obradu transakcija

Aplikacija je organizirana oko baze podataka i obrađuje korisničke zahtjeve za pristup informacijama pohranjenim u bazi ili za dodavanje novih / izmjene postojećih podataka u bazi.

Transakcija:

* Kod baza podataka to je slijed operacija koje treba obaviti prije nego promjene postanu stalne;
* Iz korisničke perspektive transakcija je bilo koji jedinstveni slijed operacija koje izvršavaju njegov cilj (naći vrijeme leta od Londona do Pariza)

Korisnici postavljaju upite (asinkrono) koje zatim obrađuje modul za upravljanje transakcijama.



**Upravljanje transakcijom**

## Primjer: organizacija bankomata



Slijed akcija potrebnih za podignuti novce s

bankomata.

## Upravljanje transakcijama



* Međusloj za upravljanje transakcijama upravlja

komunikacijom različitih vrsta terminala (bankomata

i radnih stanica), priprema podatke i šalje ih na obradu.

* Obrada upita se radi u bazi podataka i rezultat se šalje

natrag preko sloja za upravljanje transakcijama na

korisničke terminale.

## Informacijski sustavi

* Svi sustavi koji uključuju nekakvu interakciju s "zajedničkom" bazom podataka se mogu smatrati informacijskim sustavom baziranim na transakcijama.
* Informacijski sustav dozvoljava kontroliran pristup velikoj količini informacija npr. katalog biblioteke, vrijeme leta aviona, zapisi o pacijentima bolnice.
* Svi informacijski sustavi imaju istu generičku strukturu koja je organizirana u slojevima koji se sastoje od:
* Korisničkog sučelja



* Komunikacije s korisnikom
* Dohvata i izmjene informacija
* Baze podataka s modulom za upravljanje transakcijama

**Arhitektura informacijskih sustava**

## Primjer: Arhitektura LibSYS sustava

Sloj za komunikaciju s korisnikom:

* Prijavu u sustav



* Upravljanje upitima i izgledom
* Upravljanje ispisom

Sloj za dohvat podataka

* Distribuirano pretraživanje
* Dohvat dokumenta
* Upravljanje pravima
* Naplata

## Sustavi za dodjelu resursa

Ista, četvero-slojna arhitektura se može primijeniti i na drugu vrstu informacijskih sustava – sustave za dodjelu resursa.

To su sustavi koji upravljaju fiksnom količinom nekih resursa (npr. ulaznice za neki događaj, knjige u knjižari, …) i dodjeljuju ih korisnicima.

Primjeri takvih sustava su:

* Različiti rasporedi – vremenski period je resurs koji se dodjeljuje;
* Sustav biblioteke – knjige i ostali sadržaji za posudbu su resursi kojima se upravlja;
* Sustav za kontrolu zračnog prostora – zračni prostor je resurs koji se dodjeljuje.

## Arhitektura sustava za dodjelu resursa



Slojevita arhitektura koja uključuje:

* Bazu podataka resursa
* Set pravila koja opisuju kako se dodjeljuju resursi
* Alokacija resursa
* Provjera korisnika
* Upravljanje upitima
* Komponenta za isporuku resursa
* Korisničko sučelje

## Implementacija slojevitih sustava

Slojevita arhitektura se može implementirati na nekoliko načina:

* Svaki sloj se može implementirati kao odvojena komponenta, pokrenuta na odvojenom serveru. Svaki sloj ima definirana sučelja preko kojih se odvija komunikacija.
* Cijeli informacijski sustav je pokrenut na jednom računalu, tada su među-slojevi realizirani kao jedan program koji komunicira s bazom podataka.
* Implementacija sustava kroz niz odvojenih web servisa.

Korisničkom sučelju se najčešće pristupa korištenjem web preglednika.

## Arhitektura sustava za kupnju preko Interneta

Tzv. "e" sustavi su sustavi za dodjelu resursa bazirani na Internetu i prihvaćaju elektroničke naredbe za nabavku robe ili usluge.

Najčešće su organizirani u višeslojnu arhitekturu, gdje su aplikacijski slojevi međusobno povezani.



## Sustavi za obradu događaja

* Ovi sustavi reagiraju na događaje iz okoline sustava ili korisničkog sučelja.
* Osnovna karakteristika im je da sustav ne može predvidjeti kada će se događaj dogoditi i arhitektura mora biti organizirana da se događaj obradi u trenutku kada se dogodi.
* Sustavi za obradu (teksta, slika, dijagrama, …) i sustavi za rad u realnom vremenu su dvije najčešće vrste sustava baziranih na događajima. Osim njih to su:
* programi za prezentacije,
* igre,….

## Sustavi za obradu

Osnovne karakteristike:

* koristi ih jedan korisnik
* trebaju osigurati brzu reakciju na korisničke akcije
* organizirani oko dugotrajnih transakcija, pa mogu uključivati i opcije oporavka od greške.

## Arhitektura sustava za obradu



Komponente sustava za obradu teksta su objektno orijentirane:

* **Nadzor** – nadzire memoriju i detektira događaje;
* **Događaj** – prepoznaje događaje i šalje ih na obradu;
* **Naredba** – izvršava naredbe korisnika;
* **Podaci editora** – upravlja strukturom podataka editora;
* **Dodatni podaci** – upravlja drugim podacima kao

što su stil i postavke;

* **Datotečni sustav** – upravlja čitanjem i pisanjem iz / u

datoteke.

* **Prikaz** – osvježava prikaz na ekranu

## Sustavi za obradu jezika

Sustavi koji kao ulaz koriste prirodni ili umjetni jezik i generiraju neku drugi prikaz tog jezika.

Najčešći primjeri su:

* Prevodioci – prevode apstraktni programski jezik visokog nivoa u set naredbi procesora;
* Sustavi koji prevode XML opis podataka u naredbe za postavljanje upita na bazu podataka;
* Sustavi za prevođenje jednog jezika u drugi (npr. Engleski -> Hrvatski).

Koristi se u situacijama gdje je najlakše riješiti problem na način da se opiše algoritam ili podaci.

## Sustavi za obradu jezika



* **Naredbe** – opisuju što treba napraviti
* **Prevodilac** – prevodi naredbe u neki interni format koji

odgovara strojnim **naredbama strojnog jezika**.

* **Interpreter** – dohvaća strojne naredbe i izvršava ih i pri

tome koristi ulazne podatke.

* Kod većine prevodilaca interpreter je hardverska

komponenta koja obrađuje strojne naredbe.

* Kod Jave Interpreter je softverska komponenta.

***Arhitektura sustava za obradu jezika***

## Osnovne komponente

* **Leksički analizator** – uzima dijelove ulaznog jezika i pretvara ih u interni oblik.
* **Simbolička tablica** – sadrži podatke o imenima entiteta (varijablama, nazivima klasa i objekata, …) koji se koriste u tekstu koji se prevodi.
* **Analizator sintakse** – provjerava sintaksu jezika koji se prevodi (koristi definiranu gramatiku).
* **Stablo sintakse** – interna struktura koja nastaje prevođenjem programa.
* **Analizator semantike** – koristi informacije iz stabla sintakse i simboličke tablice i provjerava ispravnost ulaznog teksta.
* **Generator koda** – prolazi kroz stablo sintakse i generira strojni kod.

## Model toka podataka prevodioca



Ovaj model se koristi u gdje se programi

prevode i izvršavaju bez interakcije s korisnikom.

## Model repozitorija prevodioca



Primjer kada je sustav za obradu jezika dio

integrirane programske okoline.

# OBJEKTNO ORIJENTIRANI DIZAJN

## Objektno orijentirani proces dizajna

* Objektno orijentirani proces dizajna uključuje dizajn klasa objekata i veza među klasama.
* Kada se dizajn realizira u obliku izvršnog programa iz definicije klase se dinamički generiraju objekti.
* Strukturirani objektno orijentirani (OO) proces dizajna uključuje razvoj brojnih modela sustava.
* Razvoj i održavanje takvih modela može zahtijevati dosta truda, što za male sustave neće biti -isplativo.
* Za velike sustave, u čijem razvoju sudjeluje više različitih grupa modeli dizajna su vrlo važan mehanizam za komunikaciju.

## Procesne faze

Kako bi bilo moguće razviti sustav, tj. od početne ideje doći do detaljnog, objektno orijentiranog dizajna, treba napraviti nekoliko stvari:

* Definirati kontekst i modele korištenja sustava
* Dizajniranje arhitekture sustava
* Identifikacija najvažnijih objekata sustava
* Razvoj modela dizajna
* Specifikacija sučelja objekata

Sve ovi modeli će biti objašnjeni na primjeru meteorološke stanice smještene daleko od civilizacije, koja pohranjuje informacije o lokalnom vremenu i periodički ih šalje korištenjem satelitske veze.

## Kontekst sustava i interakcije

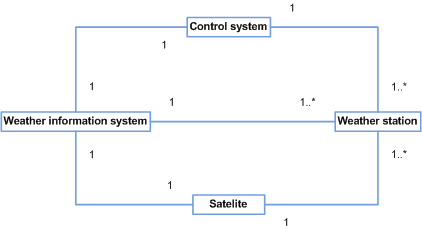
* Razumijevanje veza između softvera koji se razvija i njegove okoline je važno kako bi se moglo odlučiti na koji način osigurati tražene funkcionalnosti i kako strukturirati sustav da komunicira sa svojom okolinom.
* Razumijevanje konteksta također omogućava uspostavu granica sustava, tj. donošenje odluke koje funkcionalnosti će pružiti sam sustav, a koje neki pridruženi sustavi.

## Modeli konteksta i interakcije

* Kontekstni model je strukturni model koji prikazuje

druge sustave u okolini sustava kojeg razvijamo.

* Model interakcije je dinamički model koji prikazuje
* interakciju sustava s okolinom za vrijeme korištenja.

****

***Kontekst sustava za meteorološku stanicu Use-case dijagram meteorološke stanice***

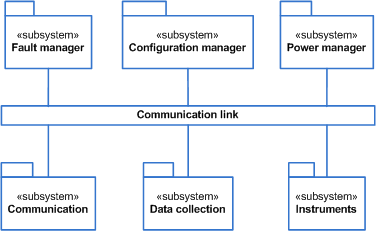
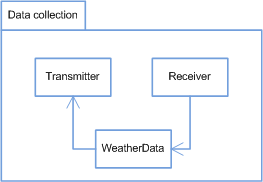
## Use case scenariji – Izvještaj o met. Uvjetima

|  |  |
| --- | --- |
| **Sustav** | **Meteorološka stanica** |
| USE CASE | Izvještaj o meteorološkim uvjetima |
| AKTORI | Meteorološki informacijski sustav, Meteorološka stanica |
| OPIS | Meteorološka stanica šalje podatke prikupljene s instrumenata u zadanom periodu Meteorološkom informacijskom sustavu. Pri tome se šalje maksimalna, minimalna i prosječna temperatura zraka i tla; maksimalni, minimalni i prosječni tlak zraka; maksimalna, minimalna i prosječna brzina vjetra; ukupna količina padalina; smjer vjetra uzorkovan u 5-minutnim intervalima. |
| POTICAJ | Meteorološki informacijski sustav uspostavlja satelitsku vezu sa stanicom i traži prijenos podataka. |
| OPIS | Traženi podaci se šalju informacijskom sustavu. |
| KOMENTARI | Upiti stanicama se najčešće šalju svako sat vremena, ali taj period nije fiksan i razlikuje se od stanice do stanice i može se mijenjati u budućnosti. |

## Dizajn arhitekture

* Kada se shvate interakcije sustava i njegove okoline, te se informacije koriste za dizajn arhitekture sustava.
* Identificiraju se osnovne komponente sustava, kao i njihove interakcije pa ih se organizira korištenjem nekog od stilova arhitekture (slojeviti, klijent-server, …)
* Meteorološka stanica se sastoji od neovisnih podsustava koji komuniciraju na način da šalju poruke preko zajedničke infrastrukture.

## Arhitektura meteorološke stanice



***Arhitektura visokog nivoa Arhitektura sustava za prikupljanje podataka***

## Identifikacija klasa objekata

* Identifikacija klasa objekata je često jedan od težih dijelova OO dizajna.
* Ne postoji jednostavna formula za identifikaciju objekata, ovisi o vještini, iskustvu dizajnera kao i o njegovom poznavanju domene.
* Identifikacija objekata je iterativan proces.

## Pristup identifikaciji objekata

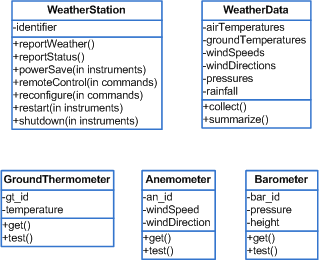
* Korištenje gramatičkog pristupa koji je baziran na opisu sustava prirodnim jezikom.
* Bazirati identifikaciju na "opipljive" stvari iz aplikacijske domene.
* Identificirati objekte prema tome u kakvom ponašanju sudjeluju.
* Koristiti analizu baziranu na scenarijima jer je iz scenarija lako prepoznati objekte, atribute i metode.

## Opis meteorološke stanice

* **Meteorološka stanica** je grupa softverski kontroliranih **instrumenata**. Ona prikuplja podatke, obavlja početnu obradu tih podataka i šalje podatke na daljnju obradu. Stanica koristi slijedeće instrumente: termometar za tlo i zrak, vjetromjer i vjetrokaz, barometar i kišomjer. Podaci se prikupljaju periodično.
* Kada stigne zahtjev za slanjem **meteoroloških podataka**, stanica obrađuje i sumira prikupljene podatke. Sumirani podaci se šalju onome tko ih je tražio.

## Klase objekata meteorološke stanice

Identifikacija objekta klasa meteorološke stanice se može bazirati na stvarnom hardveru i podacima u sustavu:



* Termometar, vjetromjer, tlakomjer
* Objekti aplikacijske domene koji predstavljaju

odgovarajući hardver (mjerne instrumente)

* Meteorološka stanica
* Osnovno sučelje meteorološke stanice prema

okolini. Odražava interakcije identificirane u

use-case modelu.

* + - Meteorološki podaci
* Sadržava obrađene podatke mjernih instrumenata.

## Modeli dizajna

* Modeli dizajna prikazuju objekte i klase objekata, kao i veze između njih.
* Predstavljaju vezu između zahtjeva i implementacije, jer trebaju biti dovoljno apstraktni kako previše detalja ne bi sakrilo veze, a moraju sadržavati dovoljno detalja kako bi ih programeri mogli koristiti u implementaciji.
* Dvije osnovne grupe modela dizajna:
* **Statički modeli -** opisuju statičku strukturu sustava preko klasa objekata i njihovih veza.
* **Dinamički modeli -** opisuju dinamičke interakcije među objektima.

## Primjeri modela dizajna

Najkorisniji modeli su:

* Modeli podsustava koji prikazuju logičko grupiranje objekata u smislene pod-sustave.
* Slijedni modeli koji prikazuju slijed interakcije objekata.
* Dijagrami prijelaza stanja (eng. state machine) koji prikazuju promjenu stanja objekata kao rezultat određenih događaja.

Drugi modeli uključuju use-case modele, modele agregacije i generalizacije.

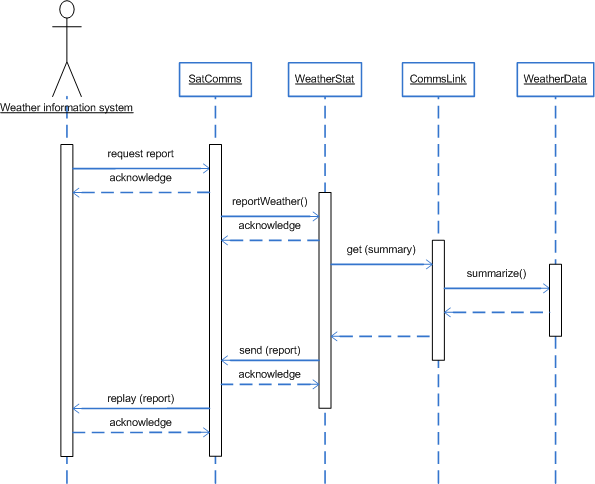
## Modeli podsustava

* Prikazuju kako je dizajn organiziran u logički povezanu grupu objekata.
* U UML-u to se prikazuje korištenjem paketa – grupiranje objekata.
* Ovo je logički model, stvarna organizacija objekata u sustavu može biti drugačija.

## Slijedni model

Slijedni model prikazuje slijed interakcije objekata

* Objekti su složeni vodoravno na vrhu dijagrama
* Vrijeme je predstavljeno okomito, pa se model čita od vrha prema dnu.
* Interakcije su predstavljene strelicama (sadrže opisni tekst), a različite vrste strelica predstavljaju i različite interakcije.
* Tanki pravokutnik na liniji života objekta predstavlja vrijeme kada je objekt kontrolni objekt u sustavu.

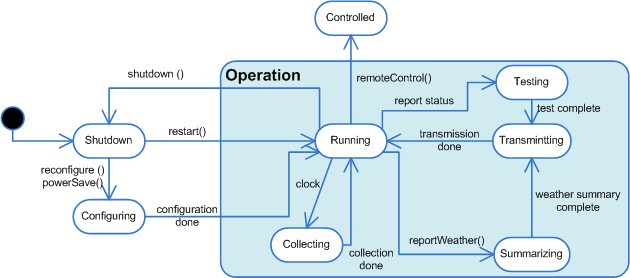


***Slijedni dijagram – opis prikupljanja podataka***

## Dijagram stanja

* Dijagram stanja pokazuje kako objekti reagiraju na različite zahtjeve za servise, te koji se prijelazi stanja događaju kao rezultat tih zahtjeva.
* Dijagram stanja je za prikaz modela višeg nivoa sustava ili prikaz ponašanja objekta u vrijeme izvršavanja.
* Najčešće nije potrebno izraditi dijagrame stanja za sve objekte u sustavu, jer je velik dio njih dovoljno jednostavan.

## Dijagram stanja meteorološke stanice



## Specifikacija sučelja

* Kako bi se moglo istovremeno raditi na razvoju različitih objekata i komponenti treba specificirati sučelja objekata.
* Objekti mogu imati nekoliko sučelja koja u biti prikazuju metode tog objekta.
* UML koristi dijagram klasa.

# Dizajn korisničkog sučelja

## Korisničko sučelje

* Korisničko sučelje bi trebalo dizajnirati na način da odgovara vještinama, iskustvu i očekivanju budućih korisnika tog sustava.
* Korisnici sustav ocjenjuju prema njegovom sučelju, a ne funkcionalnosti.
* Loše korisničko sučelje je razlog zbog kojeg se velik broj softverskih sustava nikad ne koristi.
* Loš dizajn sučelja može navesti korisnike da naprave ozbiljne greške.
* S obzirom na raspoloživi hardver, razlikujemo tekstualno i grafičko korisničko sučelje.

Osnovne karakteristike grafičkog korisničkog sučelja su:

* Veći broj prozora omogućuje istovremeni prikaz raznih informacija na istom zaslonu.
* Ikone omogućuju minimizirani slikovni prikaz datoteka, procesa ili prozora.
* Izbornici omogućuju biranje naredbi bez njihovog utipkavanja.
* Pokazivački uređaji (miš i slični) omogućuju biranje s izbornika ili pokazivanje mjesta unutar prozora koje je od interesa.
* Slike i crteži mogu se kombinirati s tekstom na istom zaslonu.

Sama aktivnost oblikovanja sučelja obično se odvija na iterativni način, u suradnji s korisnikom, korištenjem prototipova.

## U koju fazu spada definiranje k. sučelja?

Okvirni izgled sučelja se može definirati u fazi specifikacije:

* sučelje može poslužiti kao opis što sustav radi
* korisnik može odabrati sučelje u skladu s njegovim navikama i standardima.
* Strogo definiranje sučelja spada u fazu dizajna, pogotovo u slučaju složenih sustava s nestandardnim načinom komunikacije s korisnikom.

## Principi oblikovanja korisničkog sučelja

Kod oblikovanja korisničkog sučelja mora moramo uzeti u obzir fizičke i mentalne mogućnosti ljudi koji će koristiti softver.

Principi dobrog oblikovanja:

* ***Bliskost korisniku -*** Sučelje treba koristiti termine koji su bliske dotičnoj grupi ljudi, dakle jezik aplikacijske domene a ne jezik informatike.
* ***Konzistentnost -*** Slične operacije svugdje se trebaju obavljati na sličan način.
* ***Minimum iznenađenja -*** Korisnici ne smiju biti iznenađeni ponašanjem sustava ili neobaviješteni o njegovim radnjama i trebali bi biti u stanju predvidjeti ishod sličnih naredbi.
* ***Mogućnost oporavka*** - Moraju biti uključeni mehanizmi koji omogućuju korisnicima da isprave svoje greške, na primjer sustav mora tražiti od korisnika potvrdu destruktivne akcije ili mu omogućiti operaciju “undo”.
* ***Vođenje korisnika -*** Sustav mora proizvoditi poruke koje prate korisnikove akcije, te mora pružiti pomoć korisniku ovisnu o kontekstu.
* ***Uvažavanje različitosti korisnika*** - Sučelje se treba moći prilagoditi potrebama različitih vrsta korisnika, npr. početnika, povremenih korisnika, naprednih korisnika, administratora, …

## Interakcija korisnika sa sustavom

Prvi važan aspekt dizajna korisničkog sučelja je interakcija korisnika sa sustavom, tj. način na koji korisnik daje informacije i postavlja upite na sustav.

Osnovni interakcijski stilovi su:

* Direktna manipulacija
* Korištenje izbornika
* Popunjavanje formi
* Korištenja komandnog jezika
* Korištenje prirodnog jezika

Većina sustava podržava više stilova interakcije.

|  |  |  |  |
| --- | --- | --- | --- |
| **Stil interakcije** | **Prednosti** | **Mane** | **Primjeri aplikacija** |
| **Direktna manipulacija** | Brz i intuitivan.  Lako za naučiti. | Može biti teško za primijeniti.  Može se koristiti jedino kad postoji vizualna usporedba između zadatka i objekta. | Igre, CAD sustavi |
| **Korištenje izbornika** | Teže se događaju greške u radu.  Potrebno je manje tipkanja. | Prespor za iskusne korisnike.  Može postati presložen, ako ima previše izbora u izborniku. | Sustavi opće namjene. |
| **Popunjavanje formi** | Jednostavan unos podataka.  Lako za naučiti.  Lako za provjeru. | Zauzima dosta prostora na ekranu.  Javljaju se problemi kad ono što korisnik želi unijeti ne odgovara poljima u formi. | Kontrola dionica, obrada kredita, prijava poreza. |
| **Korištenje komandnog jezika** | Moćan i fleksibilan | Teško za naučiti.  Loše upravljanje greškama. | Operacijski sustavi, sustavi za nadzor i upravljanje. |
| **Korištenje prirodnog jezika** | Dostupan povremenim korisnicima.  Lako dodavanje novih naredbi. | Zahtijeva više tipkanja.  Najčešće su ovakvi sustavi nepouzdani. | Sustavi za dohvat informacija. |



***Primjer višestrukog***

***korisničkog sučelja***

## Prezentacija informacija

* Drugi važan aspekt dizajna korisničkog sučelja je prikazivanje informacija, dakle način na koji sustav korisniku predočava informacije.
* Ista informacija obično se može prikazati na tekstualni (numerički) ili se može transformirati i prikazati grafički način.
* Softver za prikaz informacije bi trebalo odvojiti od same informacije jer to omogućava prikazivanje izmjena neovisno o dijelu sustava koji proizvodi podatke.

## Prezentacija informacija

Faktori koji utječu na način prikaza informacije:

* Je li korisnika zanima precizna ili približna vrijednost?
* Je su li važne apsolutne ili relativne vrijednosti ili odnosi?
* Kako se brzo vrijednosti mijenjaju i trebaju li se promjene odmah prikazati korisniku?
* Poduzima li korisnik neku akciju kao odgovor na promjene?

## Prezentacija informacija

* Za podatke koji se ne mijenjaju za vrijeme jednog ciklusa obično se koristi tekstualni prikaz - precizan je i ne zauzima mnogo mjesta.
* Promjenjivi podaci ili podaci kod kojih je važan relativni odnosi se prikazuju grafički.





***Grafički prikaz promjenjivih podataka***

******

***Prikaz podataka o mjesečnoj Grafički prikaz relativnih podataka  
prodaji časopisa***

## Prezentacija informacija

U slučajevima kad treba prikazati jako velike količine informacija, smišljaju se posebne vizualizacije.

Primjeri:

* Informacije o vremenu skupljene iz raznih izvora prikazuju se na karti kao izobare, fronte, ciklone, anticiklone, …
* Stanje telefonske mreže prikazuje se kao skup čvorova i spojnica.
* Stanje tehnološkog procesa u kemijskoj tvornici prikazuje se tako da se pokažu pritisci i temperature u povezanom skupu spremnika.
* Skup web stranica se prikazuje kao stablo.

## Korištenje boje

* Korištenje boje daje dodatnu dimenziju sučelju i može biti dodatna pomoć korisniku da lakše shvati kompleksnu strukturu informacija.
* Boja se može koristiti za istaknuti neke izvanredne događaje.
* Najčešća greška – previše boja na ekranu.
* Preporuke:
* Limitirati korištenje boja, vodit računa o njihovom slaganju.
* Koristiti promjenu boje kako bi se naglasila promjena statusa sustava.
* Koristit promjenu boje na promišljen i konzistentan način.

## Vođenje korisnika

Treći važni aspekt izrade korisničkog sučelja je vođenje korisnika, dakle način na koji sustav prati korisnikov rad i pomaže mu.

Vođenje korisnika obavlja se na tri razine:

* ***Poruke*** koje sustav sam proizvodi kao odgovor na korisnikove akcije, dakle obavijesti, upozorenja, te poruke o grešci;
* ***Pomoć*** (eng. on-line help) koje sustav prikazuje na zahtjev korisnika;
* ***Korisnička dokumentacija*** koja se isporučuje zajedno sa sustavom.

## Vođenje korisnika – poruke

Poruke treba oblikovati uzimajući u obzir sljedeće faktore:

* ***kontekst***: sadržaj poruke prilagođava se situaciji u kojoj se korisnik nalazi;
* ***iskustvo korisnika***: opširni tekstovi za početnike, sasvim kratki za iskusne;
* ***stručni profil korisnika***: terminologija iz korisnikove struke;
* ***bonton i stil***: uljudnost, pozitivni pristup, izbjegavanje dosjetki i duhovitosti;
* ***kultura***: u skladu s običajima dotične zemlje.

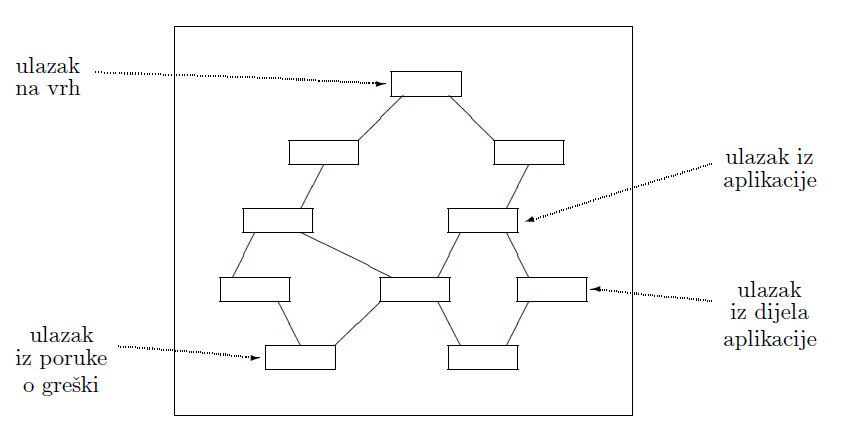


***Prikaz forme za pronalaženje podataka o Mogući odgovori aplikacije na krivo ime***

***pacijentu bolnice***

## Vođenje korisnika – pomoć

* Modul za "Pomoć" se sastoje od “kartica” teksta i međusobnih referenci.
* Obično ima složenu mrežnu strukturu.
* Sustav treba korisniku osigurati razna mjesta ulaska u ovu mrežu, ovisno o tome da li on želi općenite informacije, upute o pojedinoj aplikaciji, precizne upute o određenoj naredbi unutar aplikacije, ili pak detaljnije objašnjenje vezano uz poruku o greški.



## Vođenje korisnika – korisnička dokumentacija

Korisnička dokumentacija opisuje funkcije sustava na korisnicima razumljiv način.

Distribuira se zajedno sa sustavom na papirnatom ili elektroničkom mediju.

Sastoji se barem od slijedećih dokumenata:

* ***Funkcionalni opis -*** Namijenjen je procjeniteljima sustava i daje se grubi opis što sustav može a što ne može raditi. Priloženi su primjeri, tabele i dijagrami. Ne objašnjavaju se operativni detalji.
* ***Vodič za instalaciju -*** Namijenjen je sistem-administratoru ili korisniku. Opisuje instalaciju sustava na zadanoj vrsti računala. Sadrži opis distribucijskog medija, definiciju minimalne hardverske i softverske konfiguracije potrebne za pokretanje sustava, proceduru instaliranja i podešavanja.
* ***Uvodni priručnik -*** Namijenjen je novim korisnicima. Daje neformalni uvod u sustav, te opisuje njegovu “normalnu” upotrebu. Pisan je u formi “tečaja”, sadrži mnogo primjera, te uputa kako izbjeći uobičajene greške.
* ***Referentni priručnik -*** Namijenjen je iskusnim korisnicima. U potpunosti, te na vrlo precizan način dokumentira sve funkcije sustava, sve oblike njegove upotrebe, sve greške koje mogu nastupiti. Nije u formi tečaja. Stil je formalan, struktura je stroga. Snalaženje u tekstu osigurano je preko indeksa pojmova.
* ***Priručnik za administriranje -*** Namijenjen je sistem-administratorima ili operatorima. Opisuje aktivnosti poput stvaranja sigurnosnih kopija, upravljanja resursima, praćenja performansi, podešavanja rada, evidentiranja korisnika, postavljanja zaštite.