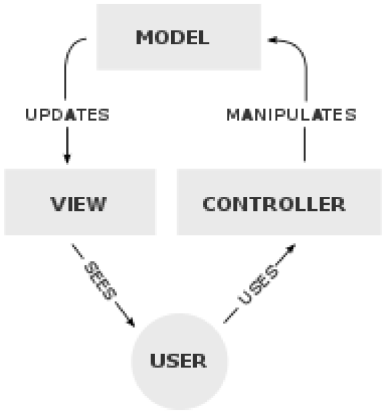
MVC arhitektura

MVC (енгл. *Model-view-controller*) arhitektura je projektni uzorak (енгл.*patern*) koji se obično koristi za razvoj korisiničkih sučelja. Počiva na ideji o ponovnoj upotrebi već postojećeg softverskog koda, olakšavanju razvoja i kasnijem održavanju aplikacionog softvera metodom razdvajanja na posebne komponente: model, prikaz podataka (pogled) i kontrolor (upravljač), pri čemu je komponenta za prikaz informacija odvojena od interakcije korisnika sa tim informacijama.



Dijagram interakcije između komponenti projektnog uzorka MVC model - prikaz - kontrolor

Садржај

Istorija

MVC arhitektura se sastoji od određenih komponenti u kojima je svaka zadužena za obavljanje specifičnih funkcija. Takvo rešenje je mnogo bolje u odnosu na ranije rešenja u kojima se sve nalazilo na jednom mestu, što je dovodilo do otežanog čitanja izvornog koda, otežanog pronalaženja i ispravljanja grešaka, kao i do znatno manje funkcionalnosti i fleksibilnosti.[1] MVC ja kao okvir softverske arhitekture nastao pre nego što je izmišljen veb-pregledač, i u početku je korišćen samo za kreiranje grafičkog korisničkog okruženja.

Norveški informatičar Trigvi Riensku (норв. *Trygve Reenskaug*) je tokom posete Ziroksu sedamdesetih godina prvi put predstavio MVC arhitekturu primenjenu na Smalltalk-76.[2][3] On je MVC kao apstraktni prikaz arhitekture softvera definisao na sledeći način:

- Model predstavlja podatke. Podaci mogu biti samo jedan objekat ili struktura objekata. Veza između modela i onoga što vidi korisnik trebala bi da bude 1 na 1. Komunikacija između modela i spoljašnjeg sveta bi trebao da se obavlja uz pomoć kontrolora.
- Pogled ili vizualna prezentacija modela, je povezana sa modelom ili jednim njegovim delom. Ponaša se kao filter, koji iz modela naglašava određene vrednosti koji opisuju podatke (atributi), dok neke druge potiskuje i tako filtrirane podatke prikazuje korisniku. Takođe, ima mogućnost ažuriranja modela, slanjem odgovarajućih zahteva korisnika.
- Kontrolor je veza između korisnika i podataka. On korisniku, u zavisnosti od njegovog zahteva prikazuje podatke.

Zatim su osamdesetih godina Džejms Altof (нем. *James Althoff*) i drugi upotrebili verziju MVC-a za Smalltalk-80 biblioteku klasa. Tek u članku iz 1988-e MVC je predstavljen kao poseban pojam.[4]

U početku MVC šablon nije funkcionisao kao danas, već je izgledao ovako[5]:

- model je podatak koji aplikacija prikazuje
- prikaz podataka iz modela, pri čemu jedan podatak može više načina prikazivanja. Na primer, očitanje temperature se može prikazati visinom stupca ili oznakom za temperaturu
- kontrolor prikuplja akcije koje korisnik napravi i na osnovu njih menja model. Tako na primer, kontrolor može da prikuplja akcije koje korisnik napravi mišem, a zatim, na osnovu njih može da izmeni model. U tom smislu, kontrolor je predstavljen kao program koji radi s ulaznim podacima, slično kao što se u korisničkom okruženju radi sa izlaznim podacima.

Na taj način, prikaz podataka se ažurira direktno, na osnovu izmena u modelu. Kada se model izmeni, on pokrene akciju koja menja prikaz podataka, pri čemu kontrolor ne upravlja prikazom podataka, već samo menja model, a prikaz podataka se ažurira tek nakon promena realizovanih u modelu.[5]

Komponente

Osnovna prednost MVC arhitekture je što se razdvajanjem u posebne celine, kod velikih projekata, na kome može da radi više osoba, omogućava laka izmena nekog elementa, bez velike intervencije u drugim elementima, kao i ponovno korišćenje već napravljenih elemenata.[6]

MVC projektni šablon se sastoji od tri zasebne, ali međusobno povezane komponente:

- **model** — je centralni deo aplikacije, koja obuhvata promenljivu (dinamičku) strukturu podataka, direktno upravljanje podacima, logikom i pravilima aplikacije
- **prikaz**, **pogled** ili **pregled podataka** (енгл. *view*) - bilo koji izlazni prikaz podataka u korisničkom okruženju (na primer grafički pomoću dijagrama), pri čemu se isti podaci mogu prikazati na više načina (na primer stubični dijagrami za menadžment i tabelarni prikaz za računovodstvo)
- **upravljač** ili **kontrolor** (енгл. *controller*) — ulazne podatke pretvara u komande koje upravljaju modelom ili prikazom podataka u korisničkom okruženju.

Praktično, u ovakvoj arhitekturi, model predstavlja „telo”, prikaz podataka su „oči”, a konotrolor „mozak” projekta.[6]

Model

Model sadrži glavne programske podatke, kao što su informacija o objektima iz baze podataka, sastoji se od skupa klasa koje modeliraju i podržavaju rešavanje problema kojim se aplikacija bavi. Obično je stabilna komponenta, koja traje koliko i sam problem. Sva poslovna logika aplikacije sadržana je u modelu.[7] Postoji nekoliko načina konstrukcije kostura modela. Naime, programer može najpre da konstruiše model, definisanjem klasa i atributa unutar klasa, te kasnije, na osnovu klasa da formira bazu podataka.[7]

Češći način je projektovanje najpre baze podataka, na osnovu kojeg se kasnije generišu klase. U svakom slučaju, potrebno je uzeti u obzir i veze između modela u aplikaciji.[7]

U MVC arhitekturi postoje model područja ili domena (енгл. *Domain model*), i model aplikacije (енгл. *Application model*). Model područja sadrži glavne podatke o strukturi podataka, dok model aplikacije sadrži sve potrebne funkcije za upravljanje tim podacima, odnosno za upis, izmenu i brisanje podataka. [7] Zavisno od potreba, razlikuju se tri vrste modela: konceptualni, logički i fizički model. Konceptualni model sadrži samo prikaz naziva entiteta. Logički model sadrži i nazive atributa, primarnih i sekundarnih ključeva. Fizički model sadrži informaciju o tome kako se podaci „vide“ u memoriji.[7]

Svi podaci se mogu dobiti preko modela, ali se model ne može direktno pozvati, već se to vrši preko kontrolora, u obliku zahteva. Ove zahtev model zatim obrađuje, te vraća kontroloru zahtevane podatke. Kontrolor dalje prikazuje dobijene podatke krajnjem korisniku.[7]

Model predstavlja jednu ili više klasa sa svojim stanjima, koja se mogu prikazati na zahtev korisničkog okruženja, a kontrolor ih može i menjati.[7]

Razlikuju se aktivni model u kome se stanje menja posredstvom kontrolora i pasivni model, kod koga se stanje u modelu vrši bez učešća kontrolora. Međutim, u praksi je češći scenario u kojem model ima aktivnu ulogu i predstavlja deo poslovne logike. U tom slučaju, model obezbeđuje metode koje će se koristiti da bi se ažuriralo trenutno stanje objekata. Model takođe može obavestiti korisničko sučelje o promeni stanja, pri čemu se prikaz trenutno osvežava.[7]

U osnovnom obliku model nema nikakvo saznanje o prikazu podataka i kontroloru, dok u proširenoj verziji MVC arhitekture, model može da sadrži i osnovne operacije za rad sa prikazima, kod takozvanog „posmatračkog" projektnog uzorka (енгл. *Observer pattern*).

Prikaz podataka

Komponenta za prikaz podataka je poslednji sloj MVC arhitekture, koji sadrži okruženje aplikacije, odnosno obezbeđuje različite načine za prezentovanje podataka dobijenih od modela, preko kontrolora.

Korisnik može videti samo ono što se vidi u ovoj komponenti, dok su model i kontrolor obično skriveni od korisnika.

Najvažnija osobina ove komponente je njena jednostavnost, jer predstavlja vizuelni prikaz modela koji sadrži metode za prikazivanje i omogućava korisniku da menja podatke. Ona svakako ne treba da bude nadležna za skladištenje podataka, osim kada se koristi keširanje kao mehanizam za poboljšanje karakteristika.

Njegov zadatak je što jasnije prikazivanje podataka koji zanimaju korisnika te do tih podataka korisnik treba doći što jednostavnijim putem. Struktura pregleda ne sme biti previše kompleksna i niti imati preveliku odgovornost. Najbolji prikazi podataka su oni koji odgovaraju samo za jedan model, te prikazuju podatke samo tog modela.[8] Moguć je prikaz jednog ili više objekata iz modela, pri čemu se ne bavi obradom podataka. Prikaz podataka zavisi od modela, odnosno metoda modela preko kojih se podaci dobijaju. Prikaz mora što jasnije prikazivati podatke koji zanimaju korisnika te do tih podataka korisnik treba doći što jednostavnijim putem.[8] Vizualna reprezentacija podataka može se nalaziti u različitim formatima, kao što su veb-pregledač, ili datoteke u HTML, XML ili PDF formatu.

Kontrolor

Kontrolor predstavlja posrednika između prikaza podataka i modela. Sadrži glavne mehanizme za kontrolu toka programa, odnosno ponašanje same aplikacije i upravlja korisničkim zahtevima.[7] On je programski najzahtevniji deo aplikacije.[8]

Kontrolor interpretira ulazne podatke korisnika i prosleđuje ih do modela ili ih prikazuje korisniku. On sadrži deo aplikacijske logike i ima sposobnost da utiče na stanje modela, odnosno odlučuje kako model treba da se izmeni, kao rezultat korisničkih zahteva, kao i način na koji će se podaci prikazati.[7]

Interakcije između komponenti

Osim podele aplikacije u tri odvojene komponente, MVC arhitektura omogućava i interakciju između njih:

- **prikaz** (енгл. *view*) - prikazuje korisniku stanje modela, obezbeđuje korisničko sučelje, pomoću koga korisnik unosi podatke i poziva odgovarajuće operacije koje treba da se izvrše nad podacima
- **kontrolor** (енгл. *controller*) - osluškuje i prihvata zahteve klijenta za izvršenje neke operacije. Nakon toga poziva operaciju koja je definisana u modelu, i ukoliko model promeni stanje, prikazuje ga korisniku.
- **model** - predstavlja stanje sistema koje se može promeniti izvršavanjem operacija nad objektima u modelu podataka. Za model nije bitno kako se upravlja podacima, niti način na koji se podaci prikazuju korisniku.

Važno je zapamtiti da prikaz u korisničkom okruženju i konotrolor zavise od modela, dok model ne zavisi niti od prikaza, niti od kontrolora komponenti, što omogućava razvoj i testiranje modela nezavisno od njegove prezentacijske logike.[9]

Prednosti i mane

Prednosti MVC arhitekture[10]:

- model se može prikazati na više načina
- lakše je dodati novi prikaz podataka (na primer novu internet stranicu koja prikazuje postojeće podatke ili deo njih)
- lakša se menja interakcija sa korisnikom
- više programera može raditi istovremeno i paralelno na različitim komponentama
- mogućnost ponovna korišćenja koda
- pojedinačni delovi se mogu lako testirati, menjati i poboljšavati
- prikaz podataka je odvojen od logike obrade podataka

Mane MVC arhitekture su[10]:

- previše kompleksna za primenu kod razvoja manjih aplikacija, što dovodi do pogoršanja kako njenog dizajna, tako i njenih karakteristika
- usled čestih izmena modela prikaz podataka može ostati preplavljen zahtevima za izmenu, što može dovesti do kašnjenja u odgovorima na zahteve.

Primena MVC arhitekture u web aplikacijama

MVC je široko prihvaćen kao projektni uzorak za aplikacije na svetskoj mreži, u svim značajnijim programskim jezicima. Nekoliko komercijalnih i nekomercijalnih programskih okvira (енгл. *framework*) koriste ovaj projektni uzorak. Svaki od njih varira u interpretaciji načina na koji su podeljene odgovornosti između klijenta i servera. Kod onih ranijih, ceo model, prikaz i konotrolor su se nalazi na serveru. Klijent je samo slao zahtev (na primer preko formulara), a potom primao kompletnu novoformiranu veb-stranicu.

U veb-aplikacijama komponenta za prikaz (pogled) sadrži: HTML, CSS, Javaskript, XML ili JSON i druge, pri čemu se prva tri ne preporučuju u kontroloru. U veb-aplikacijama kontrolor predstavlja prvi sloj koji se poziva kada veb-pretraživač pozove neku veb-adresu.

Napredak tehnologije je omogućio da se MVC komponente delom izvršavaju kod klijenta (Ajax).

Unapređivanje MVC arhitekture

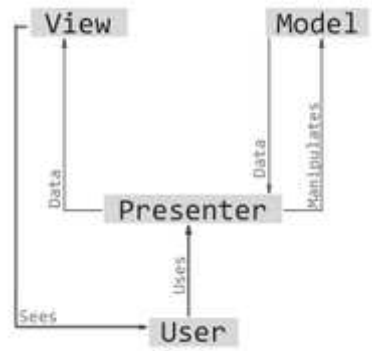
MVC arhitektura se već neko vreme bavi unapređivanjem prethodnih verzija, pa je razvijeno više različitih verzija MVC arhitekture. Sve one su uglavnom slične, s manjim razlikama u logici odvijanja procesa i nazivima komponenti.[11]

U modernim aplikacijama od 2000-te godine, kontrolor je program ili deo kompjuterskog koda koji služi za komunikaciju između modela i korisničkog okruženja, direktnim pozivanjem ili pomoću posmatrača (енгл. *observer*). S obzirom da delovi klasičnog MVC-a nemaju smisla kod „debelih" klijenata, i aspekti MVC arhitekture su takođe evoluirali, ali samo kao varijante originalnog pojma, kao što su HMVC (енгл. *Hierarchical model–view–controller*), MVA (енгл. *Model-View-Adapter*), MVP (енгл. *Model-View-Presenter arhitektura*), MVVM (енгл. *Model-View-ViewModel*), kod kojih je MVC arhitektura prilagođena na drugačije načine.

MVA

MVA arhitektura se sve više koristi u veb-tehnologiji, naročito na strani aplikacije klijenta (na primer u Javaskriptu) postoji posebna komponenta - adapter koja se nalazi između komponenti modela i prikaza podatka, koja reguliše procese tako što prati rad i procese, na osnovu odgovora koje dobija od okolnih komponenti. Na primer, kada komponenta za prezentaciju šalje „signal“ da je u tekstualno polje u formularu upisano ime, adapter prima tu informaciju i šalje zahtev modelu da ažurira podatak, koji će se kasnije spremiti u bazi podataka. Moguć je i obrnut proces. Ukoliko se podaci u modelu promene, model će poslati signal adapteru koji će reagovati u komponenti za prezentaciju korisniku, tako što će ažurirati sadržaj formulara.[11]

MVP



MVP arhitektura

MVP arhitektura je najzastupljenija kod obrasca u Androidu. Model je od korisnika najudaljeniji i sastoji se od spoljašnjih entiteta koji komuniciraju s bazom podataka ili vrše mržno povezivanje. U modelu se obavlja pretvaranje podataka iz jednog u drugi tip, kako bi se obezbedili čisti podaci u korisničkom sučelju. Sloj u kome se prikazuju podaci, odnosno sloj platforme je najduži deo MVP šablona. Služi isključivo za prikaz sučelja i podataka u njemu, kao i poslove vezane za korisničku interakciju sa slojem prezentacije. Dok ovaj sloj zavisi od platforme, ostali slojevi se mogu bez problema ponovo iskoristiti u nekim drugim projektima ili delovima aplikacije. Nakon korisničke interakcije, sloj prezentacije prima poruku o istoj. Nakon ispitivanja zahteva, iz sloja prezentacije u modelu se pokreće zahtevana akcija (kao na primer registraciju korisnika), a pri čijem se završetku, sa novim podacima prezentacija obaveštava i odatle prikazuju korisniku. Svaka akcija koju preduzima korisnik mora da bude prosleđena sloju prezentacije, koji odlučuje o tome na koji način će prikaz korisniku dalje nastaviti s radom. Ovakav šablon u kome postoji sloj prezentacije koji striktno govori sloju u kome se prikazuju podaci kako će se podaci prikazati ili šta će se naredno dogoditi, je nezgodan zbog toga što iziskuje ispisivanje veoma dugačkog programskog koda koji se ponavlja, jer je za svaki tip podatka potrebno ispisati posebnu funkciju.[12]

MPMV

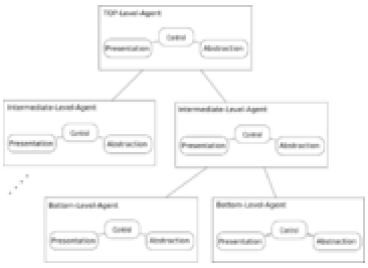

MVVMP arhitektura

MPMV arhitektura je razvijena uporedo sa MVP arhitekturom. Njena struktura je vrlo slična, s tom razlikom što se umesto sloja za prezentaciju, koristi povezivanje podataka (енгл. *Data Binding — DB*) i grupni podaci za svako stanje prikaza podataka (енгл. *View State — VS*), koji se vezuje za sloj u kome se prikazuju podaci, preko modela prikaza (енгл. *View State — VS*). Na kraju svakog ciklusa, samo se menja stanje prikaza, tako da sloj u kome se prikazuju podaci automatski dobije poruku da treba da se ažurira, odnosno nije potrebno slati zahteve, pošto sloj za prikazivanje reaguje na promene i automatski ih beleži. Međutim, pošto je MVVM arhitekturu teško primeniti, posebno kod veb-pregledača. Jedno rešenje je upotreba dodatnog sloja veb-pregledača, kojeg predstavljaju mrežni usmerivači (енгл. *router*). Drugo rešenje je korišćenje kombinacije MVP i MVVM pristupa, gdje se MVP pristup brine o porukama i veb-pregledaču, a MVVM deo se brine o toku podataka kroz ciklus i o ažuriranju sloja prikaza podataka.[12]

HMVC

HMVC arhitektura je dosta složenija struktura, u kojoj se jedna veća aplikacija razdvaja na manje, takozvne module, od kojih svaki ima sopstvenu unutrutrašnju strukturu izrađenu prema MVC arhitekturi. Najbolji primer korišćenja HMVC arhitekture su sistemi za dinamičko upravljanje sadržaja (CMS) na mrežnim stranicama. Kod njih se pojedine funkcionalnosti, kao što su mogućnost pisanja članaka ili komentara razdvojeni u module, koji funkcionišu zasebno, ali tako da korisnik ne primećuje da se stranica koja se prikazuje pomoću veb-pregledača sastoji od manjih delova.[13]

Izvori



Strukturna šema aplikacije po HMVC arhitekturi

1. Smijulj & Meštrović 2014, стр. 217.
2. Notes and Historical documents from Trygve Reenskaug, inventor of MVC.
3. "A note on DynaBook requirements", Trygve Reenskaug, 22 March 1979,SysReq.pdf .
4. Krasner, Glenn E.; Stephen T. Pope (1988). „A cookbook for using the model-view controller user interface paradigm in Smalltalk-80". *The JOT*. SIGS Publications. Also published as "A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System " (Report), ParcPlace Systems; Retrieved 2012-06-05.
5. Papratović 2016, стр. 3.
6. Markuš Mitrinović 2016, стр. 14.
7. Markuš Mitrinović 2016, стр. 17.
8. Bistrović 2018, стр. 8.
9. Markuš Mitrinović 2016, стр. 16.
10. Markuš Mitrinović 2016, стр. 19.
11. Smijulj & Meštrović 2014, стр. 218.
12. Babić 2018.
13. Papratović 2016, стр. 2.

Literatura

- Markuš Mitrinović, Elizabeta (2016). *Arhitektura MVC aplikacije sa primerima u PHP-u - Master rad* (PDF). Niš: Univerzitet u Nišu Prirodno-matematički fakultet Departman za računarske nauke. Приступљено 29. 10. 2018.
- Smijulj, Adrian; Meštrović, Ana (2014). Hirnig, Saša, yp. „Izgradnja MVC modularnog radnog okvira". *Zbornik Veleučilišta u Rijeci*. Rijeka: Veleučilište u Rijeci. **1** (2): 215—232. ISSN 1849-1723. Приступљено 30. 10. 2018.
- Babić, Filip (2018). *Arhitektura android aplikacija - završni rad*. Osijek: Sveučilište Josipa Juraja Štrosmajera. Приступљено 31. 10. 2018.
- Papratović, Nikola (2016). *Izrada aplikacije za oglasnik ponude i potražnja proizvoda - završni rad*. Osijek: Sveučilište Josipa Juraja Štrosmajera. Приступљено 31. 10. 2018.
- Bistrović, Tiana (2018). *Implementacija prototipnog sustava autentikacije putem AAI@EduHr u ASP.NET Core web aplikacijama pomoću postojećeg PHP rješenja uz korištenje baze podataka - završni rad*. Čakovec: Međimursko veleučilište u Čakovcu. Приступљено 1. 11. 2018.