UDC 004.6 INFO M: str. 35-41

PREGLED I UPOREDNA ANALIZA PREZENTACIONIH PATERNA
A REVIEW AND COMPARATIVE ANALYSIS OF PRESENTATION PATTERNS

Marko Petrović, Nina Turajlić, Ivana Dragović

REZIME: Paterni omogućavaju da se prilikom razvoja poslovnih aplikacija koriste već isprobana i proverena opšta rešenja vezana za arhitekturu, dizajn i implementaciju. Postoji mnoštvo dobro dokumentovanih i proverenih paterna koji olakšavaju rešavanje problema vezanih za prezentacioni sloj aplikacije. Razlike između samih paterna proističu iz načina podele odgovornosti vezane za funkcionalnost prezentacionog sloja u odgovarajuće komponente kao i načina međusobne interakcije ovih komponenti. U ovom radu dat je prikaz i uporedna analiza najčešće korišćenih Prezentacionih paterna, sa ciljem olakšavanja izbora odgovarajućeg paterna pri razvoju prezentacionog sloja aplikacije.
KLJUČNE REČI: Patern, prezentacioni sloj, podela odgovornosti
ABSTRACT: Patterns provide common solutions, which are tested and verified, for the architecture, design and implementation of business applications. There exist a number of well-documented and verified patterns that aid the resolving of problems inherent to the presentation layer of an application. Differences among these patterns result from the manner in which the concerns, associated with the functionality of the presentation layer, are separated into components, as well as the interaction between these components. This paper presents a review and comparative analysis of the most frequently applied Presentation Patterns, with the intention of facilitating the selection of the best-suited pattern for designing the presentation layer of an application.
KEY WORDS: Pattern, Presentation Layer, Separation of Concerns

1. UVOD

Paterni omogućavaju da se prilikom razvoja poslovnih aplikacija koriste već isprobana i proverena opšta rešenja vezana za arhitekturu, dizajn i implementaciju. Dizajn paterni pružaju pristup potvrđenim metodologijama za rešavanje različitih problema, čime omogućavaju korišćenje kolektivnog znanja i iskustva.

Princip podele odgovornosti podrazumeva da poslovna aplikacija bude tako projektovana da se njeni delovi što je moguće manje preklapaju u pogledu funkcionalnosti. Ovaj princip se najpre primenjuje pri projektovanju same aplikacije koja se u skladu sa njim razvija kao višeslojna arhitektura sastavljena od prezentacionog sloja, sloja poslovne logike, sloja za pristup podacima itd. Zatim se ovaj princip primenjuje i na pojedinačne slojeve. Osnovni zadatak većine dizajn paterna vezanih za prezentacioni sloj je jasno razdvajanje kôda koji prikazuje korisnički interfejs i prima interakcije korisnika od kôda koji se bavi logikom prezentacije. Dizajn paterni koji će biti prikazani u ovom radu daju preporuke za organizaciju prezentacionog sloja u smislu načina podele njegove odgovornosti, a međusobno se razlikuju upravo u načinu podele te odgovornosti u komponente i načinu međusobne interakcije tih komponenti.

Cilj rada je pregled paterna koji se mogu koristiti na prezentacionom sloju i njihova uporedna analiza. Stoga će najpre biti detaljnije razmatran princip podele odgovornosti, a zatim će se izložiti izabrani Prezentacioni Paterni. Na kraju će se dati uporedna analiza izloženih paterna koja bi trebalo da olakša izbor odgovarajućeg.

2. PODELA ODGOVORNOSTI (SEPARATION OF CONCERNS)

Podela odgovornosti u suštini podrazumeva podelu aplikacije na delove koji se, što je moguće manje preklapaju u pogledu funkcionalnosti. Cilj je da sistem bude projektovan na takav način da se njegove funkcije mogu optimizovati nezavisno jedna od druge, odnosno da otkaz jedne ne utiče na ostale.

Kada je u pitanju prezentacioni sloj, u opštem slučaju, aplikacija preuzima domenske podatke, prikazuje ih u korisničkom interfejsu i omogućava korisniku njihov pregled i izmenu (korisnički interfejs sadži kontrole kojima se ti podaci prikazuju). Nakon što korisnik izmeni podatke, izmene se obrađuju a zatim prenose kroz aplikaciju nazad do skladišta podataka i/ili utiču na izmenu kontrola na korisničkom interfejsu.

Prirodna je tendencija da se izvori podataka i korisnički interfejs povežu, odnosno da se kompletna definicija prezentacione logike, kojom se kontroliše opisani proces, nalazi u korisničkom interfejsu, kako bi se smanjio kôd i poboljšale performanse aplikacije. Međutim, mogući problemi koji ujedno predstavljaju i razloge za primenu Principa Podele Odgovornosti leže u sledećem:

- Kôd bi bio složen i nepregledan, težak za razumevanje, održavanje i testiranje;
- Ponavljanje istog kôda u različitim delovima aplikacije. Ovaj problem je posebno izražen kod održavanja aplikacije;
- Poslovna aplikacija često obuhvata poslovnu logika koja ne predstvalja puki prenos podataka;
- Logika prezentacionog sloja se uglavnom menja češće od poslovne logike, naročito u Web aplikacijama. Ukoliko se prezentacioni kôd i poslovna logika nalaze u istom objektu svaki put kada se izvrši i najmanja izmena korisničkog interfejsa potrebno je izmeniti objekat koji sadrži poslovnu logiku. Na ovaj način se povećava mogućnost greške i neophodno je ponovno testiranje kompletne poslovne logike;
- Ponekad je neophodno da aplikacija prikazuje iste podatke na različite načine;

- Kôd unutar korisničkog interfejsa je voma teško testirati, a kreiranje automatizovanih testova za korisnički interfejs je obično komplikovanije i zahteva više vremena nego testovi za poslovnu logiku itd.

Kako bi se izbegli ili umanjili navedeni problemi osnovna ideja Prezentacionih Paterna jeste jasna podela odgovornosti vezane za funkcionalnost prezentacionog sloja u odgovarajuće komponente. Zajednički cilj je odvajanje prezentacione funkcionalnosti od komponente koja obuhvata poslovnu logiku i logiku upravljanja podacima (Model). Međutim, dalja podela prezentacione odgovornosti na komponentu koja obuhvata korisnički interfejs i komponentu koja obuhvata prezentacionu logiku (a koja je nezavisna od konkretne implementacije korisničkog interfejsa) se može izvršiti na različite načine u zavisnost od primenjenog paterna. Dakle, Prezentacioni Paterni se međusobno razlikuju u zavisnosti od načina podele odgovornosti vezanih za prezentacionu logiku, stanja korisničkog interfejsa i sinhronizaciju, kao i načina međusobne interakcije ovih komponenti.

Osnovne odgovornosti prezentacionog sloja se odnose na:

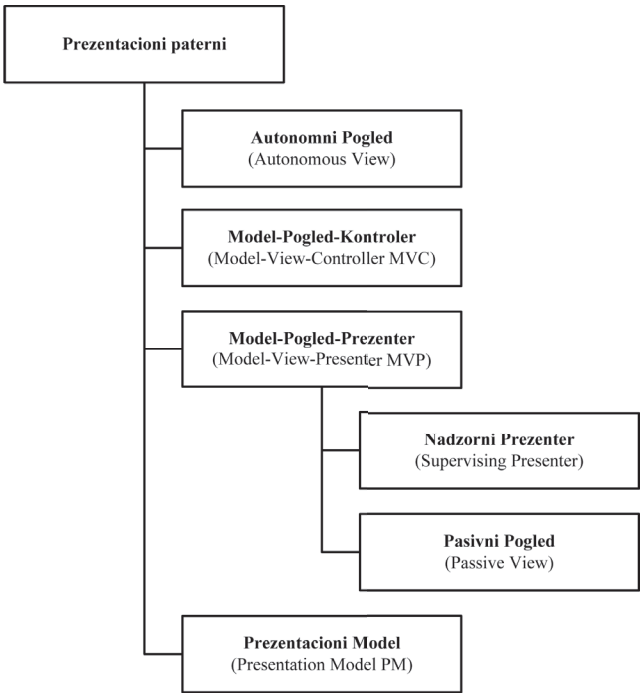
- Stanje - predstavlja tekuću sliku podataka na korisničkom intefejsu.
- Logiku - predstavlja ponašanje vezano za prikaz podataka i manipulaciju tog prikaza
- Sinhronizaciju – podaci kojie se prikazuju na korisničkom interfejsu treba da budu u skladu sa podacima u domenskom modelu. Korisnički interfejs treba da sinhronizuje podatke između elemenata prezentacionog sloja i domenskih objekata.

Prezentacioni Paterni se stoga međusobno razlikuju u zavisnosti od toga koja se od ovih odgovornosti dodeljuje kojoj komponenti.

Ono što je zajedničko svim razmatranim prezentacionim paternima je da *Model* upravlja ponašanjem, stanjima i podacima aplikacionog domena. On obuhvata podatke i poslovnu logiku, tj. aplikacionu logiku vezanu za učitavanje i upravljanje tim podacima. Model, dakle, nije samo skup podataka vezanih za neki koncept, već može implementirati veći deo aktivnosti koje se odnose na poslovna pravila datog koncepta. Između ostalog, Model može biti zadužen za validaciju, podrazumevane vrednosti, vrednosti koje su izračunavaju i slično. Model ne zna kako će biti prikazan niti ažuriran. Razlog za odvajanje Modela u zasebnu komponentu je eliminacija ponavljanja kôda i omogućavanje njegovog ponovnog korišćenja. Naime, ukoliko bi se poslovna pravila obrađivala u korisničkom interfejsu povećavala bi se verovatnoća da se isti kôd ponavlja za više različitih korisničkih interfejsa. Takođe ovako koncipiran Model podržava više različitih prikaza.

3. PREZENTACIONI PATERNI

Prezentacioni Paterni razmatrani u ovom radu prikazani su na Slici 1.



Slika 1. – *Pregled Prezentacionih Paterna*

3.1. *Autonomni pogled (Autonomous View)*

Pogled u najopštijem slučaju predstavlja skup kontrola, dok u nekim slučajevima može sadržati i ponašanje.

Patern Autonomni pogled je jedan od najjednostavnijih paterna na prezentacionom sloju. Prezentaciona logika je direktno implementirana u pogledu. Autonomni pogledi sami upravljaju svojim stanjima i komuniciraju međusobno kada je to neophodno. Dakle, i stanje i logika se nalaze u pogledu koji se implementira unutar jedne klase. U ekstremnim slučajevima, poslovna logika i komunikacija sa udaljenim servisima su takođe implementirani u pogledu.

PREDNOSTI:

- Jednostavnost.

NEDOSTACI:

- Može dovesti do problema prilikom održavanja kôda ili njegovog naknadnog tumačenja i dorade što proizilazi iz činjenice da su sve odgovornosti implementirane unutar jedne klase. Nedostatak se posebno uočava prilikom implementacije složenijih pogleda.
- Veoma otežano testiranje funkcionalnosti.

Ostali prezentacioni paterni koji će biti obrađeni u radu pokušavaju da prevaziđu ograničenja ovog pristupa. U njima je usvojena **Filozofija Skromnog pogleda (Humble View)**, koja nalaže da korisnički interfejs treba da bude što jednostavniji i da se u tom smislu logika vezana za ponašanje korisničkog interfejsa (prezentaciona logika) iz Pogleda premesti u druge, nevizuelne, klase. Skromni Pogled bi zapravo trebalo da predstavlja najuži mogući omotač oko konkretnog prezentacionog kôda. Pogled je pasivan što podrazumeva da

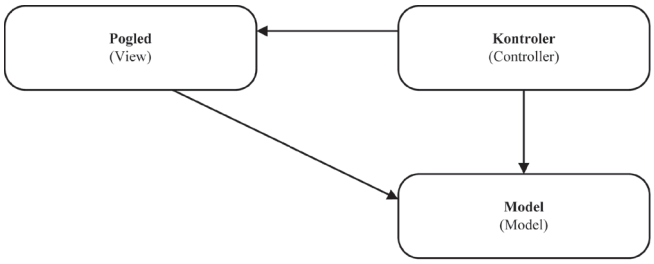
ne izvršava nikakve aktivnosti samostalno tj. bez nekog podsticaja spolja. Pogled isključivo prosleđuje događaje vezane za korisnički unos, pri čemu ih malo ili nimalo interpretira.

Prva realizacija Skromnog Pogleda je MVC arhitektura.

3.2. *Model-Pogled-Kontroler (Model View Controller MVC)*

KOMPONENTE: MVC Patern predlaže razdvajanje funkcionalnosti korisničkog interfejsa na:

- *Pogled* upravlja prikazom podataka i zapravo predstavlja vizuelnu reprezentaciju Modela. Zadužen je za renderovanje elemenata korisničkog interfejsa i trebalo bi da sadrži što je moguće manje logike.
- *Kontroler* zadužen za reagovanje na akcije korisničkog interfejsa (npr. unos podataka ili komande izdate pomoću miša ili tastature). On povezuje korisnika i aplikaciju time što omogućava interakciju između korisnika sa jedne strane i ekrana i podataka sa druge tj. interpretira unose korisnika i informiše Model i/ili Pogled o potrebnim promenama. Kontroleri su jednostavne komponente koji se bave obradom događaja vezanih za određeni Pogled i koje su zadužene za obradu i rutiranje promena.



Slika 2. – *MVC Patern*

KOLABORACIJA: (Slika 2) U MVC paternu za svaki objekat namenjen manipulaciji od strane korisnika postoji trojka Model-Pogled-Kontroler. Pogled je odgovoran za stanje, a Kontroler za logiku. Zajedno su odgovorni za interakciju sa Modelom. Svaki Pogled je povezan sa samo jednim Kontrolerom i svaki Kontroler sa samo jednim Pogledom. Kontroler i Pogled ne sadrže uzajamne reference već komuniciraju preko Modela (i Pogled i Kontrole sadrže referencu na Model). Sa druge strane, Kontroler povezuje Model i Pogled, tj. odgovoran je za sinhronizaciju. Model može biti promenjen od strane Pogleda, Kontrolera ili drugih objekata u sistemu. Kada mu se stanje promeni Model okida događaj kako bi Pogled i Kontroler bili svesni tih promena, a Pogled i Kontroler osluškuju Model. Kako korisnik unosi podatke, zahtev najpre dolazi do Kontrolera koji određuje akcije koje treba preduzeti. Neke akcije korisnika će rezultovati interakcijom sa Modelom (kao što je izmena podataka ili pokretanje metoda), dok druge mogu rezultovati vizuelnim izmenama Pogleda. Kontroler ne ažurira direktno Pogled, umesto toga on obaveštava Model i prenosi kontrolu na mehanizam osluškivanja. Obzirom da Pogled osluškuje Model on bi trebalo, u skladu sa tim, da sam sebe ažurira odnosno osveži.

Obzirom da postoji trojka Model-Pogled-Kontroler za svaki element korisničkog interfejsa, MVC patern pokazuje nedostatake u slučajevima kada ažuriranje jednog Pogleda dovodi do ažuriranja i nekog drugog Pogleda, što može dovesti do njihovog uskog povezivanja.

VARIJACIJE: [Burbeck92], Steve Burbeck opisuje dve varijacije MVC paterna:

- Pasivni model – se koristi kada jedan Kontroler ekskluzivno upravlja Modelom. Kontroler menja Model a zatim obaveštava Pogled o nastaloj izmeni i potrebi da se prikaz osveži. U ovom slučaju Model je kompletno nezavisan od Pogleda i Kontrolera, što znači da ne postoji nikakav način da prijavi izmene svog stanja.
- Aktivni model – se koristi kada Model menja stanja bez učešća Kontrolera. Ovo nastaje kada drugi izvori menjaju podatke a izmene se održavaju na Pogled. Obzirom da Model detektuje izmene u svom stanju on tada obaveštava Pogled da osveži prikaz. Međutim, jedan od osnovnih razloga korišćenja MVC paterna je da se Model učini nezavisnim od Pogleda, a ukoliko bi Model trebao da obaveštava Pogled o izmenama to bi značilo ponovno uvođenje zavisnosti. Observer patern pruža mehanizam za obaveštavanje drugih objekata o promeni stanja bez uvođenja zavisnosti.

Klasičan MVC patern se danas mahom više ne koristi u svom izvornom obliku, mada su razvijene njegove varijacije usklađene sa novim razvojnim platformama. Na primer za web aplikacije, obično se uvodi Istureni Kontroler (Front Controller) koji služi za obradu opštih infrastrukturnih problema (kao što su sigurnost, upravljanjem stanjima sesije...) i slanje zahteva pojedničnim Kontrolerima. Jedna druga varijacija ovog paterna je MVP.

3.3. *Patern Model-Pogled-Prezenter (Model View Presenter MVP)*

KOMPONENTE: MVP Patern predlaže razdvajanje funkcionalnosti korisničkog interfejsa na:

- *Pogled* koji predstavlja stukturu kontrola na korisničkom interfejsu, odnosno odgovoran je za prezentaciju i stanje. Pogled treba da sadrži što je moguće manje logike, odnosno ne treba da sadrži bilo kakvo ponašanje koje opisuje kako kontrole reaguju na akcije korisnika.
- *Prezenter* koji sadrži logiku kojom reaguje na akcije korisnika, odnosno odgovoran je za ponašanje.

KOLABORACIJA: Pogled upravlja kontrolama korisničkog interfejsa. Na akciju korisnika Pogled samo prosleđuje kontrolu Prezenteru. Prezenter zatim odlučuje kako da reaguje na tu akciju. Postoji nekoliko načina na koje Pogled može proslediti akcije korisnika Prezenteru:

- Pogled sadrži referencu na konkretan Prezenter i kada korisnik izvrši neku akciju on direktno poziva metode tog Prezentera. Ovo zahteva implementaciju dodatnih metoda u Prezenteru i rezultuje uskim povezivanjem

Pogleda sa konkretnim Prezenterom, ali je kôd čitljiviji i lakši za praćenje (u idealnom slučaju ponašanje Pogleda trebalo bi da bude jasno posmatranjem isključivo Prezentera).
- Pogled podiže događaje kada korisnik izvrši neku akciju, a Prezenter se prijavljuje na događaje. Prednost ovog pristupa u odnosu na prethodni je manji stepen povezanosti Pogleda i Prezentera.

Kada Pogled traži od Prezentera da izvrši neku akciju, to čini bez prosleđivanja bilo kakvih detalja. Stoga Prezenter, po primljenom zahtevu, treba da se obrati Pogledu i/ili Modelu i preuzme neophodne podatke.

Dalje, s obzirom da Pogled odražava stanje Model, nakon ažuriranja Modela potrebno je ažurirati i Pogled, odnosno izvršiti njihovu sinhronizaciju. Pogled i Model takođe mogu komunicirati na različite načine:

- Pogled osluškuje promene nad Modelom (Pogled je svestan Modela) i/ili,
- Prezenter osluškuje promene nad Modelom i u skladu sa njima ažurira Pogled (Pogled nije svestan Modela).

Prezenter bi zapravo, trebalo da sadrži referencu na interfejs Pogleda, a ne na njegovu konkretnu implementaciju. Ovim je omogućeno ponovno korišćenje Prezentera, odnosno veći broj Pogleda (npr. implementiranih u različitim tehnologijama) može koristiti isti Prezenter. Ovo takođe omogućava da se prilikom testiranja stvarni pogled zameni „lažnom“ implementacijom. Dakle, u idealnom slučaju Prezenter ne bi trebalo da bude čvrsto vezan sa Pogledom kako bi se Pogled mogao zameniti kompletno drugim Pogledom.

Ne postoji pravilo po kome za jedan korisnički interfejs treba da postoji po jedan Pogled i jedan Prezenter. Ako je Prezenter suviše veliki, treba ga podeliti na više klasa. Tada mogu postojati jedan nad-Prezenter i jedan nad-Pogled za ceo korisnički interfejs. Takođe ne postoji ni pravilo koje kaže da mora biti jednak broj Pogleda i Prezentera.

U odnosu na dozvoljene načine komunikacije Pogleda i Modela razlikuju se dve varijante ovog paterna:

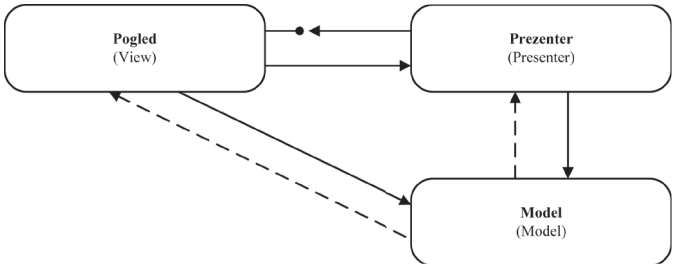
- Patern Nadzorni Prezenter - Pogled direktno interaguje sa Modelom. Pogled obuhvata logiku koja se može opisati deklarativno, dok se u složenijim slučajevima uključuje Prezenter.
- Patern Pasivni Pogled - Pogled nije svestan promena u Modelu i interakcija sa Modelom se obavlja isključivo preko Prezentera.

3.3.1. Nadzorni Prezenter (Supervising Presenter / Supervising Controller)

NAPOMENA: Obzirom da je ovaj patern varijanta MVP paterna smatrali smo da je adekvatniji naziv Nadzorni Prezenter umesto Nadzorni Kontroler mada su u literaturi prisutna oba naziva.

KOMPONENTE: Nadzorni Prezenter predlaže razdvajanje funkcionalnosti korisničkog interfejsa na:

- *Pogled* sadrži stanja i upravlja jednostavnim mapiranjem sa Modelom. Dakle, on zadržava neki stepen autonomije i kontrole koje on sadrži se i dalje mogu direktno vezati za domenske objekte.
- *Prezenter* (koji se često naziva i Kontroler) sadrži prezentacionu logiku. On ima dve osnovne odgovornosti: reagovanje na akcije korisnika i delimičnu sinhronizaciju između Pogleda i Modela. Prezenter mora osluškivati događaje koji se dešavaju na Pogledu koji mu je pridružen i kao odgovor koordinirati neophodnim ažuriranjem Pogleda i/ili Modela.



Slika 3. Patern Nadzorni Prezenter

KOLABORACIJA: (Slika 3) Kada se Model ažurira Pogled treba takođe da bude ažuriran kako bi reflektovao izmene i obrnuto, kada korisnik interaguje sa korisničkim interfejsom u Pogledu, Model treba da bude ažuriran u skladu sa tim. Kada je u pitanju sinhronizacija Pogleda i Modela:

- Pogled obično koristi neku vrstu mehanizma povezivanja (Binding) za jednostavna mapiranja, pa se u ovom slučaju promene na Modelu mogu automatski reflektovati na korisničkom interfesju bez posredstva Prezentera. Dakle, Pogled je svestan Modela i osluškuje ga. Sa druge strane, kako korisnik interaguje sa korisničkim interfejsom mogu se automatski izvršiti izmene u podacima bez posredstva Prezentera.
- Složenija logika je prepuštena Prezenteru (na primer, slučajevi dinamičkog prikazivanja, odnosno sakrivanja kontrola) tj. od Prezentera se može zahtevati da interpretira promene u Modelu kako bi ažurirao Pogled na složeniji način, a u drugim slučajevima Prezenter može ažurirati Model u ime Pogleda.

Obzirom da Pogled prosleđuje akcije korisnika Prezenteru on mora sadržati referencu na njega. Sa druge strane, Prezenter mora sadržati referencu na Pogled kako bi ga ažurirao kada se Model promeni.

S obzirom da je Nadzorni Prezenter zavisan od Pogleda, koji mu pridružen, prilikom njegovog testiranja potrebna je ili instanca konkretnog Pogleda ili objekat koji simulira ponašanje Pogleda.

NEDOSTACI:

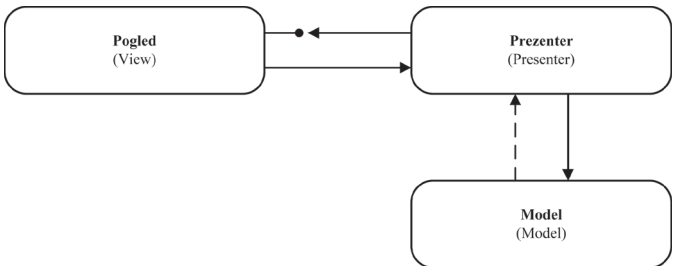
- Granicu nadzora teško je odrediti, odnosno primenu Nadzornog Kontrolera prati problem određivanja opsega Prezentera - koji deo prezentacione logike je nadležnost Prezentera. Ovo može dovesti do nekonzistntnosti na projektima, a posebno kod većih projekata.

- Kako se logika nalazi u Prezenteru on je i dalje usko povezan sa Pogledom, i mora dobro da poznaje detalje Pogleda.

3.3.2. Pasivni Pogled (Passive View)

KOMPONENTE: Pasivni Pogled takođe predlaže razdvajanje funkcionalnosti korisničkog interfejsa na:

- *Pogled* sadrži samo stanja.
- *Prezenter* sadrži svu prezentacionu logiku uključujući i mehanizam povezivanja. Pored reagovanja na akcije korisnika, on je odgovoran i za kompletnu sinhronizaciju između Pogleda i Modela. Prezenter mora osluškivati događaje koji se dešavaju na Pogledu koji mu je pridružen i kao odgovor koordinirati neophodnim ažuriranjem Pogleda i/ili Modela.



Slika 4. – *Patern Pasivni Pogled*

KOLABORACIJA: (Slika 4) Pogled je kompletno pasivan, i kao rezultat ne postoji nikakva zavisnost u bilo kom smeru između Pogleda i Modela. Prezenter ima referencu na Model. Kada Prezenter preuzme podatke iz Modela on će direktno ažurirati Pogled, čime se eliminiše potreba da Pogled ima bilo kakvo saznanje o tome kako da ispravno prikaže podatke iz objekta. Kada Pogled uputi zahtev Prezenteru da izvrši neku akciju, on to traži bez navođenja bilo kakvih konkretnih detalja pa je na Prezenteru da od Pogleda, preuzme sve informacije koje su mu neophodne.

PREDNOSTI:

- Omogućava da se testiranje fokusira na Prezenter obzirom da je Pogled pasivan.
- Mehanizam povezivanja daje najbolje rezultate kada su u pitanju nehijerarhijski organizovani objekti međutim, kada je pristutna hijerarhija ili se računaju agregirani podaci, Pasivan Pogled omogućava bolju kontrolu sinhronizacije. Alternativa je kreiranje objekata kojim se ta hijerarhijska struktura pretvara u nehijerarhijsku, ali se takvim usklađivanjem sa korisničkim interfejsom narušava struktura poslovnog domena.

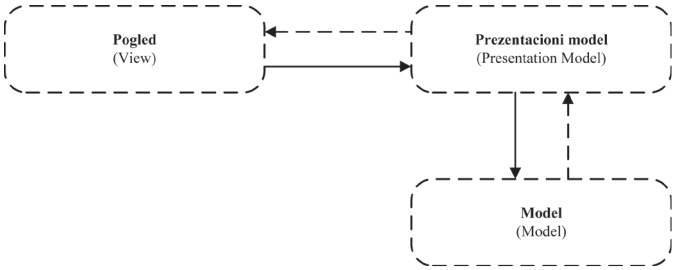
NEDOSTACI:

- Može doći do toga da Prezenter bude glomazan kao i Autonomni Pogled.
- Postoji potreba za čestom komunikacijom Pogleda i Prezentera.

3.4. *Patern Prezentacioni Model (PresentationModel PM)*

KOMPONENTE: Prezentacioni Model predlaže razdvajanje funkcionalnosti korisničkog interfejsa na:

- *Pogled* koji predstavlja prikaz na korisničkom interfejsu. Detalji vezani za korišćenu tehnologiju i grafičke komponente su unutar Pogleda.
- *Prezentacioni Model* koji sa jedne strane predstavlja apstrakciju Pogleda nezavisnu od konkretne tehnologije korisničkog interfejsa tj. predstavlja stanje i ponašanje (logiku) Pogleda ne ulazeći u način njegovog prikazivanja. Sa druge strane, Prezentacioni Model prilagođava podatake Modela u oblik lakši za prikazivanje na korisničkom interfesju tj. može se reći da vrši specijalizaciju Modela.



Slika 5. – *Patern Prezentacioni Model*

KOLABORACIJA: (Slika 5) Prema Martinu Fovleru [1], postoje dve varijante ovog paterna:

- Pogled je svestan Prezentacionog Modela - sinhronizacija je odgovornost Pogleda.
- Prezentacioni Model je svestan Pogleda - sinhronizacija je uglavnom odgovornost Prezentacionog Modela. Pogled je veoma prost, sadrži svojstva kojima izlaže svoja stanja i podiže događaje kao rezultat korisničkih akcija.

Pogled samo prikazuje stanje Prezentacionog Modela. U skladu sa promenama u Prezentacionom Modelu Pogled ažurira prikaz. To znači da je za sve odluke vezane za prikaz odgovoran Prezentacioni Model dok je Pogled prost. Prezentacioni Model jednostavno promeni svoje stanje i čeka da mehanizam povezivanja ili neki sličan koncept ažurira Pogled.

Prezentacioni Model takođe sadrži sve dinamičke informacije Pogleda ali bi trebalo da sadrži samo ona stanja Pogleda koja se mogu menjati u interakciji sa korisnikom i koja se tiču logike prezentacije. Pogled često sinhronizuje svoja stanja sa Prezentacionim Modelom (obzirom da on sadrži podatke koji su Pogledu neophodni da bi prikazao kontrole). Sinhronizacija između Prezentacionog Modela i Pogleda se obično realizuje korišćenjem Observer paterna.

Prezentacioni Model koordinira sa Modelom i predstavlja interfejs ka Pogledu čime se pojednostavljuje Pogled. Ovo znači da se promene Modela se propagiraju do Pogleda preko Prezentacionog Modela. Prezentacioni Model može komunicirati sa više domenskih objekata što znači da ne mora postajati jednoznačno preslikavanje između Prezentacionog Modela i Modela. Nekoliko Pogleda može korisititi isti Prezentacioni

Model, ali bi svaki Pogled trebalo da zahteva samo jedan Prezentacioni Model.

PREDNOSTI:

- Omogućava da se napiše logika koja je potpuno nezavisna od Pogleda koji se koristi za prikaz.
- Obzirom da detalji vezani za korišćenu tehnologiju prikaza nisu deo Prezentacionog Modela postoji mogućnost njegovog ponovnog korišćenja.

NEDOSTACI:

- Potreban je mehanizam za sinhronizaciju Pogleda sa Prezentacionim Modelom.

4. UPOREDNA ANALIZA

Sa stanovišta podele odgovornosti vezane za prezentacioni sloj, kod svih Prezentacionih Paterna Model je odgovoran za upravljanje ponašanjem, stanjima i podacima aplikacionog domena. Kod svih se stanje čuva u Pogledu, osim kod paterna Prezentacioni Model kod koga se stanje čuva u Prezentacionom Modelu. Kada je u pitanju prezentaciona logika ona se kompletno čuva u Pogledu kod Autonomnog Modela, kod MVC paterna, Nadzornog Prezentera i Prezentacionog Modela samo minimalni deo prezentacione logike se čuva u Pogledu, a ostatak u Kontroleru/ Prezenteru/Prezentacionom Modelu, dok se kod Pasivnog Pogleda kompletna prezentaciona logika čuva u Prezenteru. Kompletnu sinhronizaciju obavlja Kontroler/ Prezenter/Prezentacioni Model osim u slučaju Nadzornog Prezentera kod koga se jednostavna mapiranja obavljaju u Pogledu.

Sledeći izvor razlika među Prezentacionim Paternima proističe iz različitih načina međusobne komunikacije komponenti tog paterna. Čak su brojne varijacije i u okviru jednog konkretnog Prezentacionog Paterna. Slike prikazane u radu prikazuju najčešće načine komunikacije komponenti prisutne u literaturi.

Na kraju, prisutne su i varijacije izoženih Prezentacionih Paterna sa stanovišta same implemenatacije (koja će se klasa implementirati u kom projektu, koja će se prva kreirati, itd.)

5. ZAKLJUČAK

Postoji mnoštvo dobro dokumentovanih, detaljno analiziranih i proverenih dizajn paterna koji olakšavaju rešavanje problema vezanih za prezentacioni sloj. Mada nije uvek jednostavno odabrati odgovarajući patern, razumevanje, izbor odgovarajućeg i njegova primena rezultuje kôdom koji je lakše održavati i vodi ka boljoj modularnosti, većoj koheziji i manjem stepenu zavisnosti u aplikaciji, što su ujedno i osnovne karakteristike dobro dizajniranog sistema.

Primena Prezentacionih Paterna, koja zapravo podrazumeva odvajanje poslovne logike, prezentacione logike i korisničkog interfejsa ima sledeće prednosti:

- Kôd postaje čitljiviji i lakši za održavanje i izmenu.
- Dizajneri korisničkog interfejsa se mogu usredsrediti na vizuelne aspekte aplikacije tj. lako kreirati i menjati korisnički interfejs, dok se programeri mogu koncentrisati na logiku i strukturu aplikacije.
- Povećava se deo kôda koji se može automatizovano testirati (nezavisno od korisničkog interfejsa).
- Povećava se mogućnost ponovnog korišćenja kôda, što znači da se određeno ponašanje može koristiti u različitim delovima iste aplikacije, dok se korisnički interfejs može skrojiti po meri specifičnih uloga i lokalizacija. U svim ovim slučajevima logika je ista iako njena vizuelna prezentacija može biti sasvim drugačija, tj. omogućeno je da više Pogleda koji zahtevaju isto ponašanje dele isti kôd.
- Omogućeno je prikazivanje istih podataka u isto vreme pomoću više različitih Pogleda.
- Dodavanje novog pogleda, u opštem slučaju, ne utiče na ostatak aplikacije, što je značajno jer se zahtevi vezani za korisnički interfejs obično češće menjaju nego poslovna pravila.

Sa druge strane, primena Prezentacionih Paterna povlači i izvesne probleme:

- Paterni uvode više nivoa povezivanja čime se povećava složenost rešenja.
- Ukoliko se komunikacija zasniva na događajima debagovanje i praćenje kôda je otežano.

Ukoliko je kriterijum izbora odgovarajućeg Prezentacionog Paterna jednostavnost testiranja, treba imati u vidu sledeće. Automatizovano testiranje ponašanja kroz korisnički interfejs može biti kompleksno i zahtevati puno vremena, pri čemu je teško izolovati komponentu kod koje je nastala greška. Rizik od nastanka grešaka u Pogledu se može smanjiti izmeštanjem većeg dela, ako ne i sve logike iz Pogleda u ostale komponente koje je lakše testirati. Dakle, kôd samog Pogleda bi trebalo da bude što jednostavniji. Ukoliko je bitno automatizovano testiranje korisničkog intefejsa tada su MVP patern i patern Prezentacioni Model bolji od MVC paterna. Kod Prezentacionog Modela odnosno Pasivnog Pogleda testiranjem Prezentacionog Modela odnosno Prezentera, respektivno, testira se veći deo rizika vezanog za korisnički interfejs bez potrebe testiranja elemenata samog korisničkog interfejsa, obzirom da je kompletna prezentaciona logika smeštena u njima. Kod Prezentacionog Modela jedini preostali rizik je u mapiranju. Sve akcije korisnika i sva logika prikaza preusmeravaju na Prezentacioni Model pa kontrole u Pogledu treba samo da definišu način svog mapiranja u njega. Pasivni Pogled elimiše čak i najmanju rizik prisutan kod Prezentacionog Modela, jer Prezenter odlučuje kako da reaguje na akcije korisnika i popunjava kontrole podacima. Pogled ne sadži nikakvo ponašanje čak ni mapiranje. Kod Pasivnog Pogleda cena je što je potreban test dubler da imitira ekran prilikom testiranja. Slična je i situacija kod Nadzornog Prezentera. Prisustvo jednostavnih mapiranja uvodi izvestan stepen rizika ali uz prednost što se ona mogu deklarativno specificirati. Testiranje Nadzornog Prezentera otežano je činjenicom da je Prezenter svestan pogleda. Prednost Pasivnog Pogleda u odnosu na Nadzorni Prezenter i Prezentacioni Model je u

tome što obe alternative zahtevaju da Pogled obavlja neki deo sinhronizacije što rezultuje ponašanjem koje je teško testirati.

Izbor odgovarajućeg Prezentacionog Paterna zavisi i od prirode poslovne aplikacije koja se razvija. Kada se isti Pogledi mogu koristiti sa različitim podacima MVC je dobar izbor. A kada su u pitanju kompleksni prikazi, sa puno interakcije sa korisnikom, MVC patern može biti komplikovan za implementaciju obzirom da će za svaku interakciju postojati zaseban Kontroler. MVP patern je, u tom slučaju, bolji pošto se ta složena logika može učauriti u jednu klasu i nezavisno testirati. Za realizaciju Prezentera kod paterna Nadzorni Prezenter je potrebno manje kôda (implementacija je jednostavnija) što proizilazi iz činjenice da on implementira samo deo prezentacione logike. Ako je potrebno da aplikacija koristi različite tehnologije za korisnički interfejs i MVP i Prezentacioni Model pružaju podršku za više korisničkih interfejsa. MVP Patern je preporučljivo koristiti kada podaci koji se prikazuju podržavaju mehanizme povezivanja i ne zahtevaju dodatnu konverziju ili prilagođavanje pre prikaza. Sa druge strane Prezentacioni Model ima prednost kada je potrebno izvršiti konverziju ili prilagođavanje podataka pre prikaza, kao i kada je potrebna izvršiti validaciju podataka pre ažuriranja korisničkog interfejsa. I na kraju, neke tehnologije koje obezbeđuju automatizovanu izgradnju arhitekture zahtevaju korišćenje određenih paterna.

6. REFERENTNA LITERATURA

[1] Fowler M.: "Patterns of Enterprise Application Architecture", The Addison-Wesley, 2002.
[2] http://msdn.microsoft.com, MSDN Library
[3] http://www.jeremydmiller.com/ppatterns
[4] Burbeck S.: "Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)."University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive. Dostupno na http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.
[5] http://www.dotnetfunda.com/articles/article830-comparison-of-architecture-presentation-patterns-mvpscmvppvpmmvvm-an-.aspx
[6] http://www.aspiringcraftsman.com/2007/08/interactive-application-architecture/

Marko Petrović, Fakultet organizacionih nauka Univerziteta u Beogradu,
marko.petrovic@fon.rs
Oblasti interesovanja: Projektovanje informacionih sistema, Modelovanje poslovnih procesa, Razvoj zasnovan na modelima.

Nina Turajlić, Fakultet organizacionih nauka Univerziteta u Beogradu,
nina.turajlic@fon.rs
Oblasti interesovanja: Projektovanje informacionih sistema, Razvoj zasnovan na modelima, Modelovanje poslovnih procesa, Metode optimizacije.

Ivana Dragović, Fakultet organizacionih nauka Univerziteta u Beogradu,
ivana.dragovic@fon.rs
Oblasti interesovanja: Projektovanje informacionih sistema, Razvoj zasnovan na modelima, Upravljanje sistemima, Fuzzy Logika.