Projektovanje softvera

Projektni uzorci

Uvod

- Materijali pripremljeni prema knjizi:
  Gama, Helm, Johnson, Vlissides, *Design Patterns*

- Christopher Alexander govori o uzorcima u građevinskoj arhitekturi:
  - "Svaki uzorak opisuje neki problem koji se ponavlja u našem okruženju i tada opisuje jezgro rešenja tog problema na takav način da to rešenje može da se koristi milion puta, a da se ne izvede ni dva puta na isti način"
- Definicija je primenljiva i na uzorke u objektno-orijentisanim sistemima

- Projektni uzorak predstavlja zabeleženo široko primenljivo iskustvo u projektovanju sistema
- Termini: projektni uzorak, uzor, obrazac, šablon, modla, mustra (engl. *design pattern*)

O OO projektnim uzorcima

- OO projektni uzorci su opisi komunicirajućih objekata i njihovih klasa koji su prilagođeni da reše neki opšti projektni problem u posebnom kontekstu

- Karakteristike OO projektnog uzorka:
  - identifikuje učestvujuće klase i objekte, njihove relacije, njihove uloge u saradnji i raspodelu odgovornosti
  - sistematično imenuje, objašnjava i ocenjuje uopšteno projektno rešenje ponavljajućeg problema u OO sistemima
  - predstavlja ponovo upotrebljivo OO projektno rešenje

- Svaki projektni uzorak ima 4 bitna elementa:
  - naziv uzorka
  - postavku problema
  - opis rešenja
  - diskusiju posledica

Naziv uzorka

- Koristi se da asocira na projektni problem, njegovo rešenje i posledice primene pomoću par reči
- Imenovanje uzorka proširuje projektni rečnik
  - omogućava projektovanje na višem nivou apstrakcije
  - pojednostavljuje komunikaciju u timu
  - olakšava dokumentovanje projekta
- Ponekad se u literaturi sreće i više imena za isti uzorak

Postavka problema

- Objašnjava opšti problem
- Navodi motivaciju za primenu uzorka
  - na primeru u nekom posebnom kontekstu
- Opisuje:
  - uopšten projektni problem,
    npr. kako reprezentovati algoritme kao objekte
  - strukture klasa ili objekata
    simptomatične za nefleksibilan dizajn
  - uslove koji moraju da budu ispunjeni za primenu uzorka

Opis rešenja

- Opisuje elemente koji predstavljaju jezgro rešenja
  - njihove uloge, relacije, odgovornosti i saradnje
- Rešenje predstavlja apstraktni opis
  načina organizovanja elemenata (klasa i/ili objekata)
- Rešenje ne opisuje konkretan dizajn ili implementaciju
- Rešenje treba da posluži kao šablon
  - da može da se primeni u konkretnim slučajevima

Posledice

- Posledice su rezultati primene uzorka
- Bitne su za razmatranje projektnih alternativa i razumevanje cene i dobiti primenom uzorka
- Posledice se često odnose na vreme i prostor
- Ponekad se odnose i na jezik i implementacione stvari
- Posledice najčešće uključuju uticaj na:
  - prilagodljivost (fleksibilnost) sistema
  - proširivost (ekstenzibilnost) sistema
  - prenosivost (portabilnost) sistema

Primer primene uzoraka: MVC

- MVC je arhitekturni okvir za razvoj aplikacija
  - skraćeni naziv koji potiče od trijade klasa: *Model*, *View* i *Controller*
- Poreklo – realizacija korisničkih interfejsa u *Smalltalk* jeziku
- *Model* – konkretan objekat „poslovne logike“
- *View* – objekat prikaza objekta poslovne logike u sloju GUI
- *Controller* – objekat opisuje reakciju prikaza na ulaz korisnika
- Posmatra se ponašanje (kroz dve interakcije objekata):
  - *Model-View*
  - *View-Controller*
- Posmatra se strukturiranje (kroz hijerarhiju objekata):
  - *View*
- Primeri za 3 projektna uzorka

Model – *View* interakcija (1)

- Prikaz mora da reflektuje aktuelno stanje modela
- Kad se podaci modela (objekta poslovne logike) promene
  - model signalizira promenu prikazima koji zavise od njega
  - svaki prikaz dobija priliku da se ažurira
- Omogućeno više uzajamno nezavisnih pogleda na model
  - svaki pogled – prikaz posebnog aspekta modela
- Jednostavno mogu da se dodaju novi prikazi
  - ne menja se model
  - ne menjaju se drugi prikazi
- Primer:
  - model sadrži neke statističke podatke
  - postoje tri prikaza: tabelarni, histogram, pita

Model – *View* interakcija (2)

- Protokol između klasa *View* i *Model* je tzv. *subscribe-notify*
  - prikazi se pretplaćuju kod modela na obaveštavanje
  - model obaveštava pretplaćene prikaze kada se dogodi promena
  - kada dobiju obaveštenje (signal) prikazi čitaju novo stanje modela
  - prikazi prilagođavaju svoj izgled novom stanju modela
- Generalizacija problema obaveštavanja "pretplatnika"
  - promene jednog objekta treba da utiču na proizvoljan broj drugih
  - nema potrebe da menjani objekat zna za detalje drugih
- Rešenje generalnog problema
  - opisano uzorkom Posmatrač (*Observer*)
  - uloge: subjekat (MVC *Model*) i posmatrač (MVC *View*)

Komponovanje prikaza (*View*)

- Jedna od mogućnosti MVC arhitekture
  - prikazi (*Views*) mogu da budu ugnežđeni
- Primer:
  - kontrolni panel je prikaz koji sadrži ugnežđene prikaze dugmadi
- Klasa *CompositeView* izvedena iz *View*
  - predstavlja vrstu prikaza koji sadrži druge prikaze
  - može da se pojavi gde god se očekuje objekat osnovne klase *View*
  - dobija se objektna hijerarhija (stablo) prikaza
- Generalizacija problema strukturno složenih objekata (sklopova)
  - grupisanje objekata gde je grupa istog (nad)tipa kao i pojedini objekat
- Rešenje generalnog problema
  - opisano uzorkom Sastav (Sklop, Kompozicija, engl. *Composite*)
- Sličan je odnos klasa *Container* i *Component* u Javi

View – *Controller* interakcija (1)

- MVC omogućava promenu načina reakcije *View* na ulaz korisnika
- Primer:
  - moguće je redefinisati odgovore na događaje sa tastature/miša
  - vršiti istu obradu pritiska na ekransko dugme, taster i stavku menija
- MVC kapsulira mehanizam odgovora u objekat tipa *Controller*
- Postoji hijerarhija klasa kontrolera koja olakšava kreiranje novog
  - novi kontroler se kreira kao varijacija nekog postojećeg
- *View* agregira i koristi objekat potklase *Controller*
  - objekat kontrolera specificira strategiju odgovora
- Za promenu strategije odgovora
  - samo se zamenjuje objekat kontrolera drugom vrstom kontrolera
- Ekransko dugme i stavka menija mogu da koriste isti kontroler

View – *Controller* interakcija (2)

- Izmenu strategije je moguće vršiti i u vreme izvršenja
- Primer:
  - objektu klase *View* može da se onemogući interakcija (*disabled* stanje) tako što mu se dodeli kontroler koji ignoriše ulaze
- Generalizacija problema promenljivog algoritma
  - statička ili dinamička izmena algoritma
  - algoritam definiše ponašanje nekog objekta (konteksta)
- Rešenje generalnog problema
  - opisano uzorkom Strategija (*Strategy*)
  - objekti strategije apstrahuju i kapsuliraju algoritam ponašanja
  - lako se zamenjuju u nekom kontekstu (promena pokazivača)

Klasifikacija projektnih uzoraka

- Namena i nivo apstrakcije opisa varira kod različitih uzoraka
- Kao i svaka druga klasifikacija, ova klasifikacija
  - doprinosi boljem i bržem snalaženju pri traženju odgovarajućeg uzorka
  - usmerava napore ka otkrivanju novih uzoraka
- Klasifikacija koristi dva kriterijuma za razvrstavanje uzoraka
  - kriterijum namene
    - šta opisuje uzorak: stvaranje, strukturu ili ponašanje
  - kriterijum domena, odnosno nivoa apstrakcije opisa
    - na koji nivo apstrakcije se odnosi uzorak: klasni ili objektni
      - klasni – nivo (domen) apstrakcija, objektni – nivo (domen) primeraka apstrakcija
    - pojam „domen“ se ne odnosi na domen primene (aplikativni domen) uzorka
- Klasifikacija nije strogo formalna, pa tako ni sasvim precizna
  - zasniva se u dobroj meri na intuitivnoj proceni
  - ipak, pomaže razumevanju prirode projektnog uzorka

Kriterijum namene

- Kriterijum deli uzorke prema nameni
  - odražava čime se uzorak bavi (šta opisuje?)
- Namena uzorka može da bude da opiše
  - stvaranje (*creational patterns*)
    - uzorci opisuju stvaranje (proizvodnju) objekata
  - strukturu (*structural patterns*)
    - uzorci opisuju kompozicije objekata ili klasa
  - ponašanje (*behavioral patterns*)
    - uzorci opisuju načine interakcije objekata ili klasa

Kriterijum domena

- Kriterijum deli uzorke prema nivou apstrakcije opisa
  - odražava nivo apstrakcije opisa koji koristi uzorak (kako opisuje?)
  - da li se uzorak fokusira na apstrakcije (klasni domen) ili na primerke apstrakcije (objektni domen)
- Klasni uzorci (*class patterns*)
  - fokusiraju se na relacije između klasa i potklasa
  - ove relacije su generalizacije/specijalizacije
  - one su statičke – fiksirane u vreme prevođenja
- Objektni uzorci (*object patterns*)
  - fokusiraju se na relacije između objekata
  - ove relacije su uglavnom primerci asocijacija (veze)
  - one su dinamičke – mogu da se menjaju u vreme izvršenja
- Većina uzoraka je u objektnom domenu

Karakteristike vrsta uzoraka

- Klasni uzorci kreiranja
  - klase delegiraju stvaranje objekata (ili njihovih delova) potklasama
- Objektni uzorci kreiranja
  - neki objekti delegiraju stvaranje objekata (ili njihovih delova) drugim objektima
- Klasni uzorci strukturiranja (strukture)
  - komponuju klase kroz speicjalizaciju i koriste nasleđivanje implementacije
- Objektni uzorci strukturiranja (strukture)
  - opisuju načine asembliranja (sklapanja celina od nekih) objekata
- Klasni uzorci ponašanja
  - komponuju klase kroz specijalizaciju da kompletiraju algoritme/tok kontrole
- Objektni uzorci ponašanja
  - opisuju kako grupa objekata sarađuje da obavi neki zadatak

Prostor projektnih uzoraka

- Tabela prikazuje klasifikaciju uzoraka po dva kriterijuma
  - kolone sadrže vrste uzoraka po nameni
  - vrste sadrže vrste uzoraka po domenu

		Namena		
		uzorci stvaranja	**uzorci strukture**	**uzorci ponašanja**
Domen	**klasni uzorci**	Fabrički metod	Adapter (klasni)	Šablonski metod Interpreter
	objektni uzorci	Apstraktna fabrika Graditelj Prototip Unikat	Adapter (objektni) Most Sastav Dekorater Fasada Muva Zastupnik	Lanac odgovornosti Komanda Iterator Posrednik Podsetnik Posmatrač Stanje Strategija Posetilac

Odnosi između uzoraka

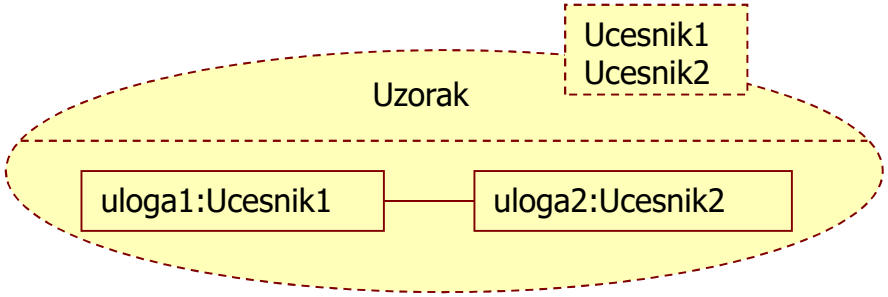
- Postoje i drugi načini za organizovanje uzoraka:
  - neki uzorci se često koriste zajedno
    - npr. Sastav (*Composite*) i Iterator (*Iterator*) ili Posetilac (*Visitor*)
  - neki uzorci su alternative jedni drugima
    - npr. Prototip (*Prototype*) i Fabrički metod (*Factory Method*)
  - neki uzorci različitih namena imaju sličnu klasnu strukturu
    - npr. Sastav (*Composite*) i Dekorater (*Decorator*)
  - neki uzorci različitih namena imaju sličnu objektnu strukturu
    - npr. objektni Adapter i Dekorater (*Decorator*)

Katalog projektnih uzoraka

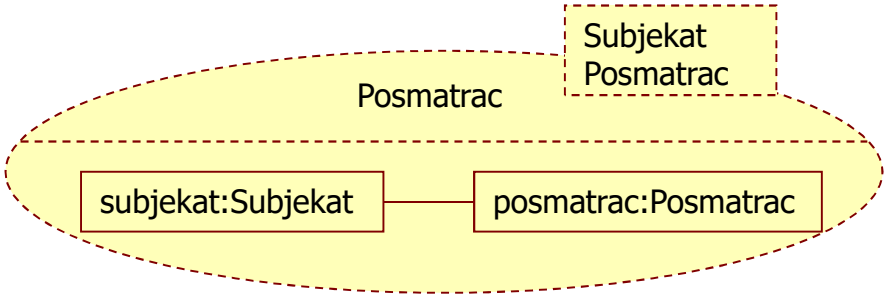
- Katalog (u knjizi *Design Patterns* - GoF) sadrži za svaki uzorak sledeće stavke:
  - ime uzorka i klasifikacija
  - namena – odgovori na pitanja "šta radi? čemu služi? koji problem rešava?"
  - drugi nazivi – isti uzorak može da ima više naziva
  - motivacija – scenario koji ilustruje projektni problem i loš dizajn rešenja
  - primenljivost – situacije u kojima uzorak treba da se primeni (da se izbegne loš dizajn)
  - struktura – klasni (eventualno i objektni) dijagram koji opisuje uzorak
  - učesnici – klase i objekti koji učestvuju u uzorku i njihove odgovornosti
  - saradnje – kako učesnici sarađuju da ostvare svoje odgovornosti (eventualno d. interakcije)
  - posledice – diskusija dobrih i loših strana primene uzorka
  - implementacija – zamke, preporuke i tehnike kojih treba biti svesan pri implementaciji
  - primer koda – fragment koda koji ilustruje kako uzorak može da se implementira
  - poznate primene – primeri primene uzorka u realnim sistemima (barem po dva)
  - povezani uzorci – koji uzorci su bliski sa datim, koje su razlike, sa kojima se često koristi
- Nema u katalogu:
  - UML notacija – grafički simbol za saradnju koja realizuje uzorak

Definicija uzoraka u UML notaciji

- Definicija uzorka (strukturirana parametrizovana saradnja):

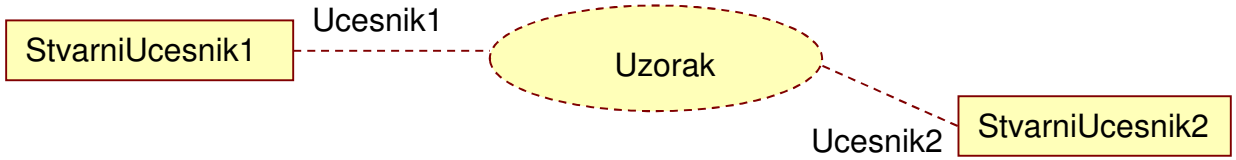


- Formalni parametri parametrizovane saradnje:
  - tipovi uloga učesnika u definiciji projektnog uzorka
- Primer:



Primena uzoraka u UML notaciji

- Primena uzorka:



- Argumenti parametrizovane saradnje:
  - tipovi stvarnih učesnika – konkretne klase u modelu
- Relacija između saradnje i stvarnih učesnika je vezivanje (*binding*)
  - isprekidana linija na kojoj se navodi formalni tip učesnika iz definicije
- Primer: