RAZVOJ APLIKACIJA KORIŠTENJEM UZORAKA DIZAJNA

APPLICATION DEVELOPMENT USING DESIGN PATTERNS

Mario Konecki, Zlatko Stapić, Tihomir Orehovački

SAŽETAK

Dinamika poslovnog svijeta i ostalih aspekata društva zahtijeva brz razvoj novih aplikacijskih rješenja za potporu poslovanju. Razvoj aplikacija se mora svesti na svoj vremenski minimum a mogućnosti krivih načina oblikovanja i razvoja programske podrške se moraju eliminirati jer dovode do nepotrebnog trošenja vremena i odgađanja isporuke samog proizvoda. Uzorci dizajna predstavljaju jedan korak u rješavanju navedene situacije i postaju danas sve popularniji. Uzorci dizajna predstavljaju prepoznata rješenja za ponavljajuće probleme u području dizajna aplikacijske podrške na način da osiguravaju poznat i prokušan način rješavanja pojedinog problema. Mogu se grupirati i klasificirati prema namjeni i načinu primjene. U ovom radu bit će dan pregled svih grupa i vrsta uzoraka dizajna zajedno s mogućnostima njihovog korištenja te problemima koji se mogu pojaviti.

Ključne riječi: uzorak dizajna, ponavljajući problem, razvoj, aplikacijska podrška

ABSTRACT

Dynamics of business world and the rest aspects of human society demand rapid development of new application solutions for business support. Application development has to be brought to its time-consumption minimum and the possibility of using wrong ways of design and development of applications has to be eliminated because it consumes time for no useful results and it delays the final product delivery. Design patterns are one of the steps in solving this situation and they become more and more popular. Design patterns are general solutions for reoccurring problems in software design and in this way they provide known and tested way to solve some particular problem. They can be grouped and classified according to their purpose and the way of usage. In this paper an overview of all groups and types of patterns along with the possibilities of their usage is given along with the problems that can occur in their usage.

Keywords: design pattern, reoccurring problem, development, application

1. UVOD

Poslovni svijet danas je postao jako zahtjevan u pogledu velikog broja aplikacija za potporu poslovnim procesima. Potreba je proces razvoja ubrzati i u što većoj mjeri automatizirati. Jedan od mogućih pristupa rješavanju dijela ovog problema je i korištenje uzoraka dizajna.

Uzorak dizajna je opće i ponovo iskoristivo rješenje nekog problema koji je opće poznat u području oblikovanja programske podrške. To je predložak koji govori kako riješiti neki određeni problem i može biti primjenjiv u više različitih situacija. Općeg je karaktera, što znači da ne može biti izravno preveden u programski kod. On definira objekte, veze, interakcije, itd. koje vode nekom određenom rješenje problema. Osim uzoraka dizajna u području programske podrške postoje i druge vrste uzoraka kao što su npr. arhitektonski uzorci, međutim uzorci dizajna se bave isključivo problemima oblikovanja programske podrške i kao takvi su predmet proučavanja informatičke struke. Općenito, možemo reći da su uzorci dizajna rješenja za ponavljajuće problema u nekom određenom kontekstu.

2. ASPEKTI PROGRAMSKE PODRŠKE

Pri oblikovanju programske podrške postoji puno aspekata koji se moraju uzeti u obzir i pomoću kojih odgovarajući uzorak dizajna može biti lakše pronađen. Neki od ovih aspekata su:

- **Ekstenzivnost** (Extensibility) – dodavanje novih mogućnosti bez većih promjena u arhitekturi
- **Robusnost** (Robustness) – otpornost na teške uvjete rada, nekontrolirani unos, nisku memoriju i druge neočekivane uvjete
- **Pouzdanost** (Reliability) – mogućnost izvođenja zadanih funkcija unutar zadanih uvjeta za zadani vremenski period
- **Otpornost na greške** (Fault-tolerance) – otpornost na teške uvjete i mogućnost oporavka od uvjeta u kojima dolazi do grešaka i otkazivanja ispravne funkcionalnosti komponenti
- **Sigurnosti** (Security) – mogućnost odgovora i reakcije na maliciozne radnje
- **Održivost** (Maintainability) – laka izmjena i nadogradnje u svrhu očuvanja postojeće funkcionalnosti
- **Kompatibilnost** (Compatibility) – mogućnost rada i komuniciranja sa proizvodima drugih proizvođača
- **Modularnost** (Modularity) – programska podrška je izgrađena od dobro definiranih i nezavisnih komponenti (modula) što pomaže lakšem održavanju. Komponente mogu biti implementirane i testirane zasebno prije integracije što omogućava podjelu posla unutar razvojnog projekta.
- **Ponovna iskoristivost** (reusability) – ponovna iskoristivost programskih komponenti

3. IDEJA O UZORCIMA DIZAJNA

Sama ideja uzoraka je u početku bila arhitektonski koncept koji je osmislio Christoper Alexander (1977-1979). Kasnije su Kent Beck i Ward Cunningham počeli iskorištavati ideju korištenja uzoraka u području programiranja. Njihova ideja je prezentirana nešto

kasnije na godišnjoj ACM konferenciji OOPSLA [3][4]. 1994. GoF (Gang of Four – Banda četvero) - Erich Gamma, Richard Helm, Ralph Johnson, i John Vlissides napisali su knjigu naslova Design Patterns: Elements of Reusable Object-Oriented Software [2] koja je imala veliki utjecaj na popularnost uzoraka dizajna.

Uzorci dizajna predstavljaju testirana i pouzdana rješenja problema koji su uobičajeni i eliminiraju mogućnost nekih većih problema koji se mogu pojaviti kada programer oblikuje program prema svom nahođenju bez ikakvih smjernica. Korištenjem uzoraka dizajna olakšava se razumijevanje programskog koda svima koji su upoznati s konkretnim uzorkom koji je korišten.

4. DOKUMENTIRANJE UZORAKA DIZAJNA

Uzorci dizajna su opisani u dokumentaciji koji ih čini lakima za korištenje. Ona opisuje kontekst u kojem se uzorak koristi, elemente i veze unutar konteksta s kojima uzorak radi i koje pokušava razriješiti i također neke praktične sugestije za konkretna rješenja. Ova dokumentacija se sastoji od nekoliko sekcija. One nisu uvijek iste ali jedan od poznatijih formata ove dokumentacije je onaj opisan u knjizi Design Patterns: Elements of Reusable Object-Oriented Software [2] koja se već spomenuta. Od posebnog značaja su sekcije Struktura (Structure), Učesnici (Participants) i kolaboracija (Collaboration) koji opisuju prototip mikro-arhitekture koju programeri kopiraju i prilagođavaju željenom dizajnu svog programa da bi riješili neki problem koji je opisan odabranim uzorkom dizajna.

Sekcije ovog formata dokumentacije su [2]:

- Ime uzorka dizajna i klasifikacija: opisno i jedinstveno ime koje pomađe u identifikaciji uzorka
- Namjera: opis cilja uzorka i razlog njegovog korištenja
- Također znan kao: drugi nazivi uzorka
- Motivacija: scenarij koji se sastoji od problema i konteksta unutar kojeg uzorak može biti korišten
- Primjenjivost: sitacije u kojima je uzorak primjenjiv; kontekst uzorka
- Struktura: grafička reprezentacija uzorka (Dijagram klasa ili interakcije je često korišten)
- Učesnici: lista klasa i objekata korištenih u uzorku te njihove uloge u dizajnu
- Kolaboracija: Opis kako su klase i objekti korišteni u uzoraku u interackciji jedni s drugima
- Posljedice: opis rezultati, usputnih efekata i posljedica korištenja uzorka
- Implementacija: opis implementacije uzorka
- Primjer koda: ilustracija kako uzorak može biti korišten u programskom jeziku
- Poznate primjene: primjeri stvarnog korištenja uzorka
- Povezani uzorci: drugi uzorci koji su na neki način povezani s navedenim uzorkom; rasprava o sličnostima s drugim sličnim uzorcima

5. KRITIKE UZORAKA DIZAJNA

Usprkos mnogim prednostima postoje i neke kritike uzoraka dizajna.

Neke od kritika i problema su:

- Uključivanje dodatnih razina indirekcije
- Za razliku od komponenata uzorci ne pružaju povnovu iskoristivnost
- Neki tvrde da se uzorci dizajne ne razlikuju značajno od drugih oblika apstrakcije
- itd.

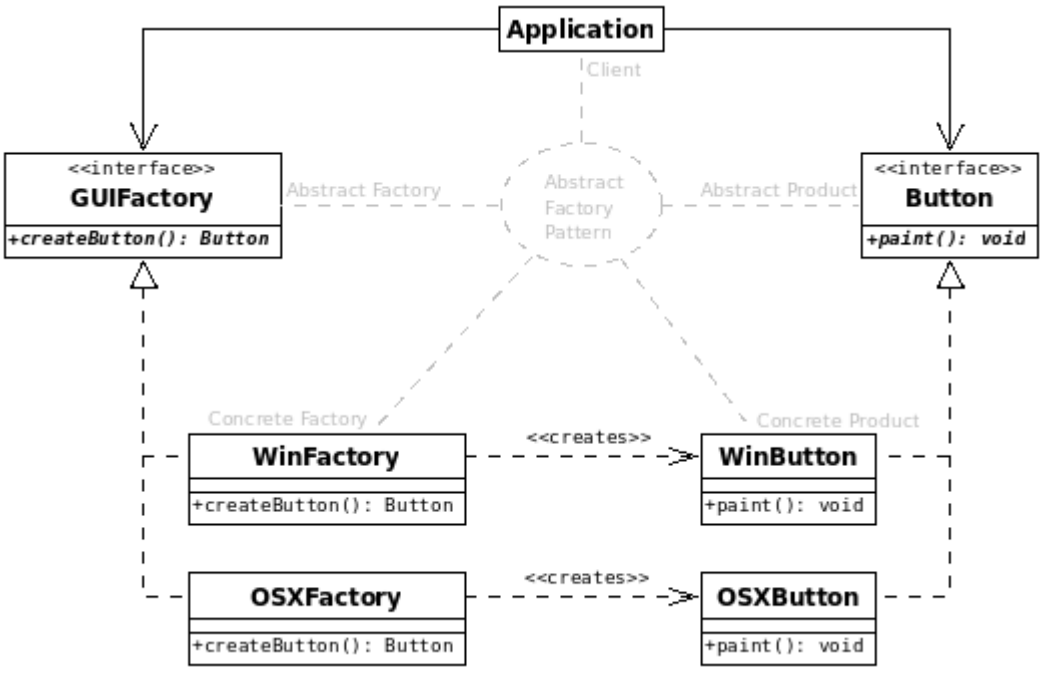
6. KLASIFIKACIJA UZORAKA DIZAJNA

Uzorci dizajna se klasificiraju prema određenim karakteristikama uzoraka što znači da se neki uzorak ovisno o svojim karakteristikama može naći u više grupa.

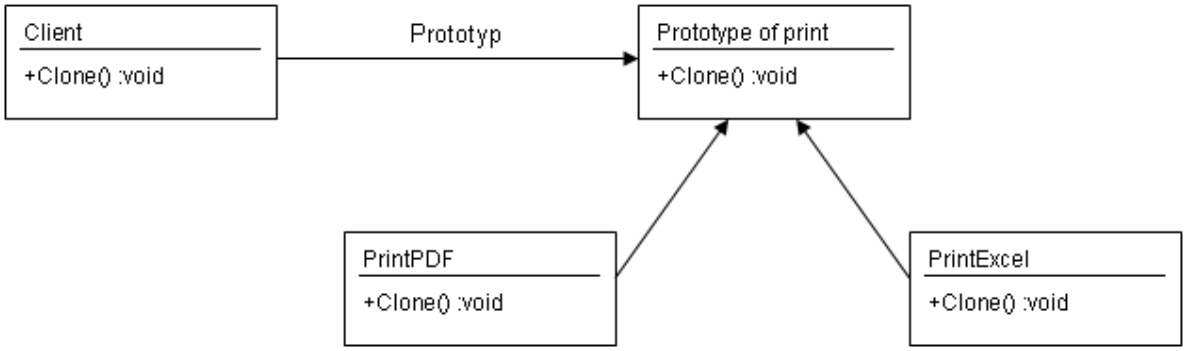
Glavna ideja klasifikacija je da se željeni uzorak pronađe brže i lakše.

Postoji mnogo kriterija po kojima se uzorci mogu klasificirati i postoji mnogo klasifikacije. U ovom radu ćemo spomenut najklasičniju od njih a to je ona dana u knjizi Design Patterns: Elements of Reusable Object-Oriented Software design patterns [2] ili takozvana GoF klasifikacija gdje su uzorci dizajna grupirani u 3 grupe:

- **Stvaralački uzorci (Creational patterns)**
- **Strukurni uzorci (Structural patterns)**



Slika 1: Primjer korištenja Abstract factory uzorka



Slika 2: Primjer korištenja Prototype uzorka

- **Uzorci ponašanja (Behavioral patterns)**

Ova klasifikacija dijeli uzorke prema njihovoj namjeni.

Stvaralački uzorci su uzorci koji se bave mehanizmima stvaranja objekata, pokušavajući stvoriti objekte na način prikladan za datu situaciju.

Strukturni uzorci su uzorci koji pružaju jednostavan način realizacije veza između entiteta.

Uzorci ponašanja su uzorci koji identificiraju i pomažu u realizaciji komunikacije između objekata.

Za potrebe ovog rada spomenut će se samo neki predstavnici svake spomenute grupe uzoraka dizajna [1].

Stvaralački uzorci:

1. Abstract factory
2. Prototype
3. Singleton

Strukturni uzorci:

1. Adapter
2. Bridge
3. Composite

Uzorci ponašanja:

1. Interpreter
2. Observer

6.1 Abstract factory

Abstract factory uzorak pruža sučelje za kreiranje porodice povezanih ili međusobno ovisnih objekata bez specificiranje nijihovih konkretnih klasa. Ovaj uzorak pruža mogućnost enkapsulacije grupe individualnih factories-a (isto jedan od stvaralačih uzoraka) koji pripadaju istoj porodici.

Korištenje ovog uzorka je objašnjeno na slici 1 koristeći UML [5]:

Klijent (Application) stvara konkretne implementacije abstract factory-ja (GUIFactory) i onda koristi generička sučelja da stvori objekte koji pripadaju porodici objekata koje abstract factory enkapsulira. Abstract factory derivira mnoge konkretne klase (factories) za sve moguće objekte koji mogu biti stvoreni. Konkretni objekti (products) su stvoreni i koriste se od strane klijenta koristeći samo generičko sučelje. Na ovaj način klijent ne mora znati ništa o implementaciji zatraženog konkretnog objekta i nije ga briga za ništa osim za korištenje objekta.

6.2 Prototype

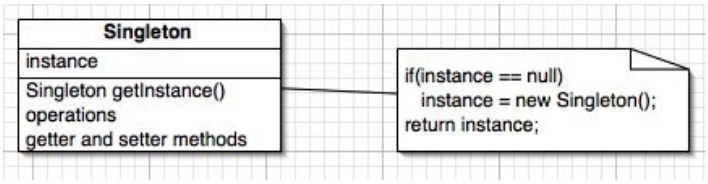
Ovaj uzorak definira vrstu objekata za stvaranje koristeći prototipsku instancu i stvara nove objekte kopiranjem prototipa.

Korištenje ovog uzorka je objašnjeno na slici 2.

Klijent, umjesto da instancira konkretne klase, jednostavno poziva clone() operaciju (factory metoda) dajući pri tome podatke o željenoj deriviranoj klasi što rezultira u stvaranju željene konkretne klase kopiranjem prototipa i kreiranjem nove instance.

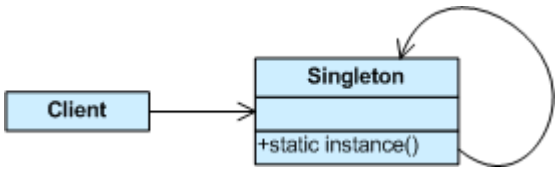
6.3 Singleton

Singleton se koristi u svrhu restrikcije instantaciranja klase na samo jedan objekt. On osigurava da klasa ima samo jedan objekt i pruža globalnu točku pristupa tom objektu.



Slika 3: Primjer korištenja Singleton uzorka [6]

Singleton održava statičku referencu na jedinu singleton instancu i vraća referencu na tu instancu iz statičke metode instance().



Slika 4: Primjer korištenja Abstract factory uzorka [1]

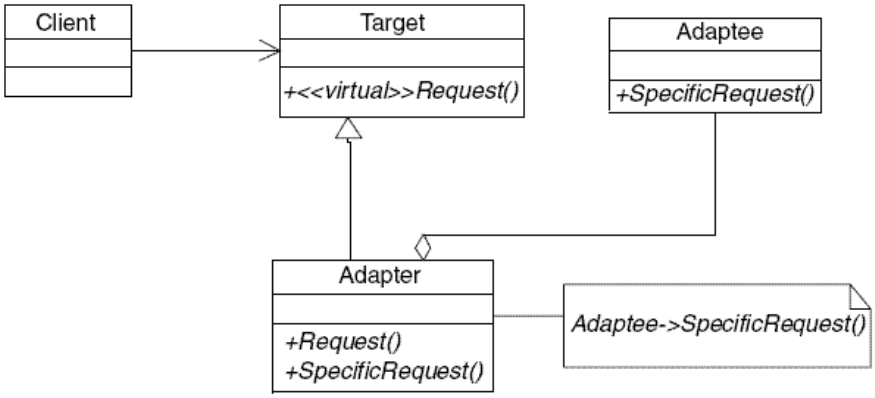
Dakle, u osnovi jedna instance je odgovorana za sve radnje i ona je opće dostupna preko globalne točke pristupa.

6.4 Adapter

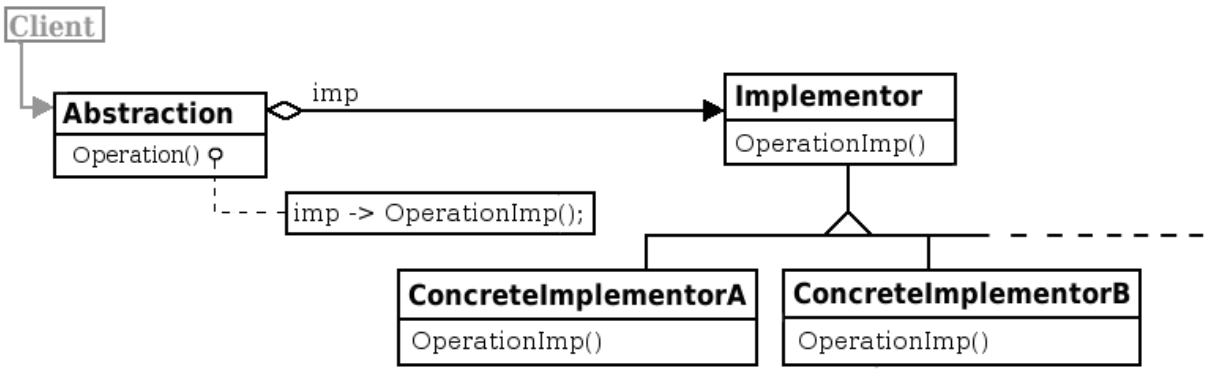
Ovaj uzorak konvertira sučelje jedne klase u drugo sučelje koje klijent očekuje. Adapter dopušta klasama koje ne bi bez adaptera nikako mogle funkcionirati zajedno zbog nekompatibilnih sučelja da rade zajedno. Adapter funkcionira kao omotač ili modifikator postojeće klase. On ruža drugačiji tj. prevedeni pogled klase.

Ovaj uzorak dizajna je koristan u situacijama gdje već postojeća klasa pruža neke ili sve servise koje klijent treba ali ne koristi sučelje koje klijent treba.

Primjer na slici 5 ilustrira ovu primjenu:



Slika 5: Primjer korištenja Adapter uzorka [7]



Slika 6: Primjer korištenja Bridge uzorka

6.5 Bridge

Odvaja apstrakciju od njezine implementacije tako da obje mogu varirati nezavisno [8].

Client
: objekt koji koristi bridge uzorak dizajna

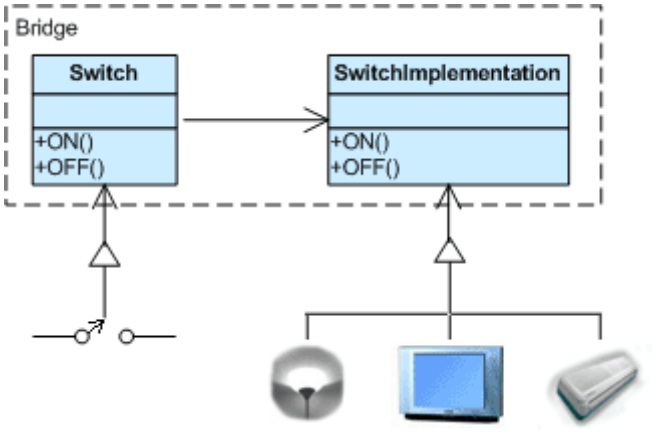
Abstraction
: definira abstraktno sučelje
: održava referencu prema Implementatoru

Implementor
: definira sučelje za implementacijske klase

ConcreteImplementor
: implementira sučelje Implementora

Još jedan primjer ovog uzorka dizajna je dan na slici 7:

U ovom primjeru prekidač kontrolira razne uređaje na način da ih uključuje ili isključunje. Funkcionalnost prekidača je jasna ali on može biti realiziran tj. implementiran na različite načine – kao klasičan prekidač, kao senzor pokreta, kao poluga, itd.



Slika 7: Primjer korištenja Bridge uzorka [1]

6.6 Composite

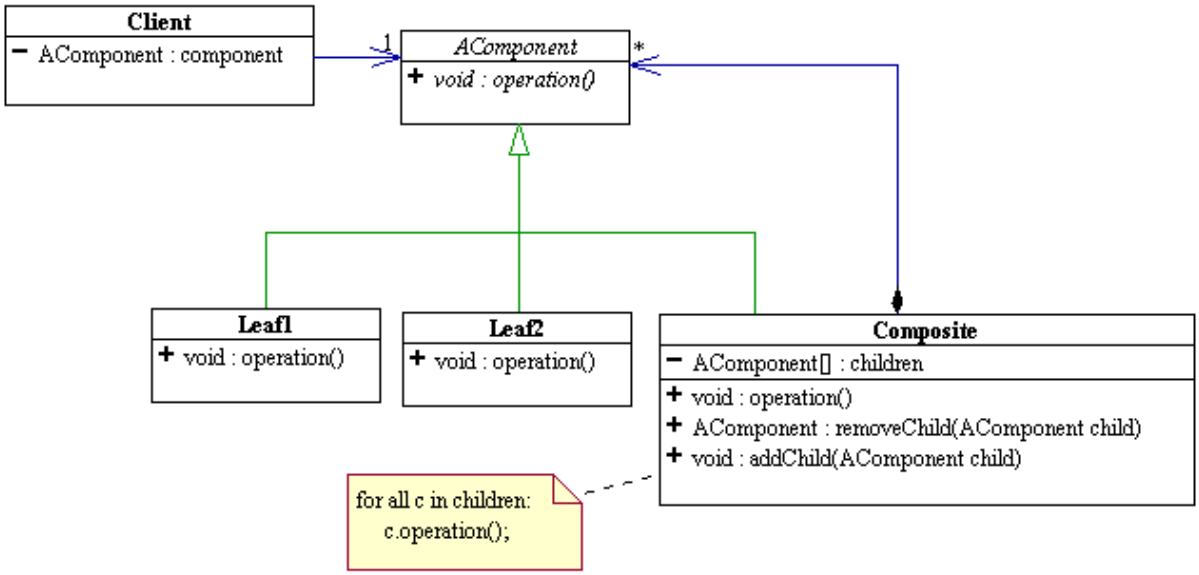
Komponira objekte u strukturu drva da bi reprezentirao cjelina-dio hijerarhiju. Composite dopušta klijentu da tretira individualne objekte i kompozicije objekata uniformno.

Composite klasa je konkretna komponenta poput Leaf1 i Leaf2 ali nema operation() dio tj. vlastito ponašanje. Umjesto toga, Composite je sačinjen od kolekcije drugih apstraktnih komponenti, koje mogu biti tipa bilo koje druge konkretne komponente uključujući i Composite. Unificirajuća činjenica je da su svi oni apstraktne AComponents. Kada operation() metoda Composite objekta biva pozvana, ona jednostavno prosljeđuje zahtjev sekvencijalno svim svojim komponentama-djeci i možda i sama obavi nešto obrade. Npr., Composite objekt može sadržavati referencu prema Leaf1 i Leaf2 instancama. Ako klijent sadržava referencu prema tom Composite objektu i pozove njegovu operation() metodu, Composite objekt će prvo pozvati operaciju na svojoj Leaf1 instanci i onda operation() na svojoj Leaf2 instanci. Združeno (composite) ponašanje Leaf1 plus Leaf2 je postignuto bez dupliciranja koda.

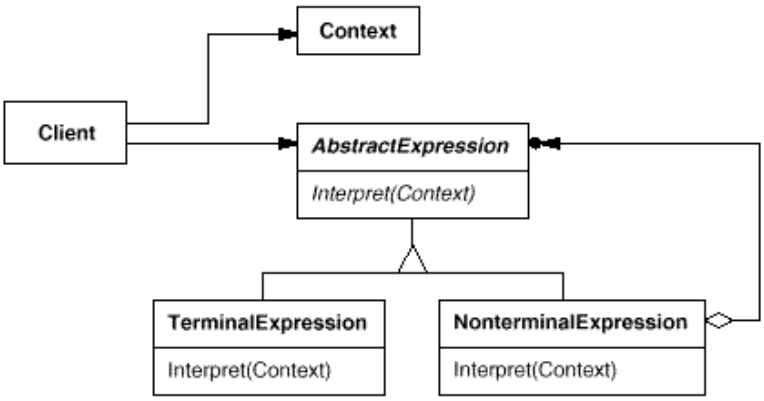
6.7 Interpreter

Uz zadan jezik, definira reprezentaciju za svoju gramatiku uz interpreter koji koristi reprezentaciju da interpretira rečenice jezika. Specijalizirani jezici često omogućuju da se neki problem riješi puno prije nego u jezicima opće namjene i tu svoju primjenu nalazi ovaj uzorak.

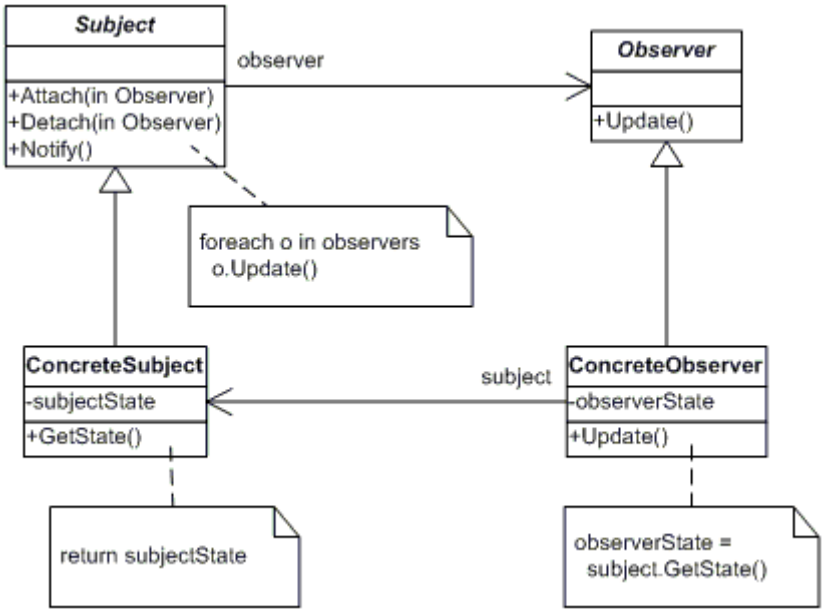
Interpreter sugerira modeliranje domene sa rekurzivnom gramatikom. Svako pravilo u gramatici je ili composite (pravilo koje referencira druga pravila) ili terminal (list čvor u strukturi drva). Interpreter se oslanja na rekurzivnu granu stabala Composite uzorka za interpretaciju rečenica koje je tražen da procesira.



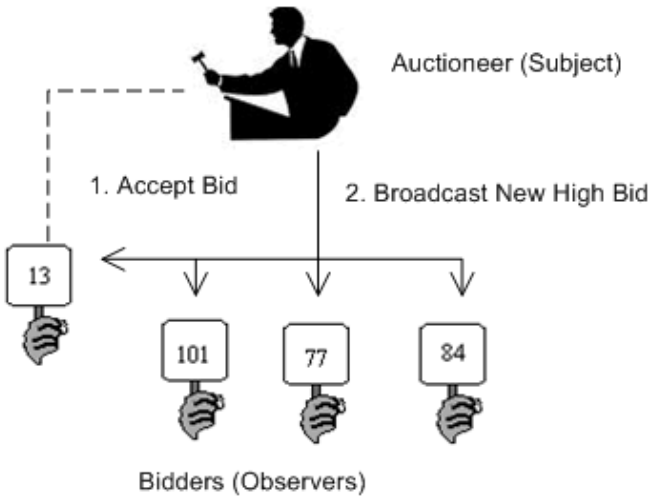
Slika 8: Primjer korištenja Composite uzorka [9]



Slika 9: Primjer korištenja Interpreter uzorka [10]



Slika 10. Primjer korištenja Interpreter uzorka [11]



Slika 11: Primjer korištenja Interpreter uzorka [1]

6.8 Observer

Definira jedan-na-više ovisnost između objekata tako da kada jedan objekt promjeni stanje, svi koji su povezani s njim i ovise o njemu su obaviješteni i ažurirani automatski. Ovaj uzorak dizajna se većinom koristi za implementaciju distribuiranih sistema za upravljenje događajima.

Primjere ovog uzorak dizajna možemo vidjeti na slikama 10. i 11.:

U ovom primjeru voditelj aukcije promatra učesnike u aukciji te s obzirom na njihove reakcije prihvaća novu cijenu i to obznanjuje ostalima koji na to dalje reagiraju daljnim podizanjem cijene na isti način sve do završetka aukcije tj. prodaje predmeta.

7. ZAKLJUČAK

Dinamika života i poslovanja se neprestano povečava i programska potpora postoje neophodan dio sve većeg sustava rada. Nužno je prepoznati načine ubrzanja razvoja ovakve podrške i načine stvaranja novog na jasne i prepoznatljive načine. Uzorci dizajna su jedan od korak ka tome cilju pružajući projektantu i programeru jasne okvire za rješavanje preopznatljivih problema čime olakšavaju i ubrzavaju cjelokupan razvojni proces.

Literatura:

1 Shvets, A., Design Patterns Simply
2 Gamma, E., Helm, R., Johnson, R., Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
3 Reid, S., Panel on design methodology, OOPSLA, 1987.
4 Beck, K., Cunningham, W., Using Pattern Languages for Object-Oriented Program, OOPSLA, 1987.
5 http://www.lepus.org.uk/ref/companion/AbstractFactory.xml, učitano 10.5.2008.
6 http://faculty.washington.edu/stepp/courses/2005winter/tcss360/readings/12a-singleton.html, učitano 10.5.2008.
7 http://wiki.forum.nokia.com/index.php/Design_Patterns_in_Symbian, učitano 10.5.2008.
8 Shalloway, T., Design Patterns Explained: A New Perspective on Object-Oriented Design
9 http://www.exciton.cs.rice.edu/javaresources/DesignPatterns/composite.htm, učitno 10.5.2008.
10 http://www.cs.mcgill.ca/~hv/classes/CS400/01.hchen/doc/interpreter/interpreter.html, učitano 10.5.2008.
11 http://www.codeproject.com/KB/scripting/Observer_Pattern_JS.aspx?print=true, učitano 10.5.2008.

Podaci o autorima:

Mario Konecki, dipl. inf.
e-mail: mkonecki@foi.hr

Zlatko Stapić, dipl. inf.
e-mail: zlatko.stapic@foi.hr

Tihomir Orehovački, dipl. inf.
e-mail: tihomir.orehovacki@foi.hr

Fakultet organizacije i informatike
Sveučilište u Zagrebu
Pavlinska 2
42 000 Varaždin
tel. +385 42 390 800
fax. +385 42 213 413

Mario Konecki je diplomirao na Fakultetu organizacije i informatike u Varaždinu 2005 godine. Tijekom studija dva puta je nagrađen kao najbolji student na godini. Za vrijeme studija bio je demonstrator na kolegijima "Programiranje I", 'Operacijski sustavi" i "Sustavi temeljeni na znanju". Tijekom dvije posljednje godine studija počeo je raditi na većim projektima kako za Fakultet organizacije i informatike, tako i za razna poduzeća. Po završetku fakulteta radio je jedan semestar kao asistent na Tekstilno-tehnološkom fakultetu u Varaždinu na kolegiju "Računalstvo". Od 3. mjeseca 2006. godine radi na Fakultetu organizacije i informatike u Varaždinu u zvanju asistenta na kolegiju "Programiranje I".

Zlatko Stapić je od 2006. godine asistent na Katedri za razvoj informacijskih sustava na Fakultetu organizacije i informatike u Varaždinu, te polaznik poslijediplomskog doktorskog studija Informacijske znanosti na istom fakultetu. Kroz studij, te

tijekom dosadašnjeg kratkog radnog iskustva dobio je više od deset različitih nagrada i priznanja, uključujući nagradu za najboljeg studenta, Rektorovu nagradu, nagradu „Zlatna Arca“ za inovativnost te nekoliko Dekanovih nagrada. Zlatko je do sada sudjelovao u nekoliko znanstvenih i stručnih projekata, pohađao je seminare i radionice u svrhu proširenja praktičnih znanja, te je objavljivao znanstvene i stručne radove u području razvoja programskih proizvoda, rudarenja podataka i sigurnosti, što su mu ujedno i područja od primarnog interesa. Zlatkov detaljnjiji životopis, s drugim važnim podacima može se pronaći na njegovoj osobnoj web stranici, na http://www.foi.hr/nastavnici/stapic.zlatko/index.html.

Tihomir Orehovački diplomirao je 2005. godine na Fakultetu organizacije i informatike u Varaždinu, smjer Informacijski sustavi. Nakon diplomiranja, kraće je vrijeme radio kao nastavnik u osnovnoj i srednjoj školi. Od 2006. godine radi na Fakultetu organizacije i informatike u zvanju asistenta te sudjeluje u izvođenju nastave na predmetima Programiranje I i Strukture podataka. Praktično informatičko znanje nadopunjavao je nizom seminara i radionica. Područja od posebnog interesa su mu web tehnologije, e-obrazovanje, umjetna inteligencija, upravljanje znanjem, generativno programiranje i simulacijsko modeliranje te je iz tih područja objavio desetak znanstvenih i stručnih radova. Polaznik je poslijediplomskog sveučilišnog doktorskog studija na matičnom fakultetu.