PRIMER ODRŽIVE SOFTVERSKE STRUKTURE KORIŠĆENJEM ABSTRACT FACTORY PATERNA
EXAMPLE OF SUSTAINABLE SOFTWARE STRUCTURE USING ABSTRACT FACTORY PATTERN

Ana Korunović[1], Siniša Vlajić[2]
[1]Fakultet organizacionih nauka, ana.korunovic@fon.bg.ac.rs
[2] Fakultet organizacionih nauka, sinisa.vlajic@fon.bg.ac.rs

Apstrakt: *Predmet istraživanja ovog rada predstavlja primena softverskih paterna u kreiranju održive strukture softverskog sistema. Istraživanjem je bilo potrebno uvideti koja mesta u programu mogu dovesti do eksponencijalnog nagomilavanja programskog koda i do pojave haosa prilikom dodavanja novih korisničkih zahteva. Na tim mestima je bilo neophodno primeniti koncept paterna i sprečiti kreiranje neodržive i neefikasne strukture softverskog sistema. U radu je objašnjen koncept paterna projektovanja, prednosti njihove upotrebe u razvoju softverskih sistema, kao i struktura rešenja Abstract Factory paterna. U studijskom primeru pokazano je kako se može kreirati održiva arhitektura softverskog sistema korišćenjem koncepta paterna, na primeru integracije četiri heterogene Javine tehnologije grafičkog korisničkog interfejsa (Swing, JavaFX, SWT i Apache Pivot). Struktura koja je dobijena predstavlja osnovu za efikasnu nadogradnju i razvoj.*
Ključne reči: *Softverski paterni, Abstract Factory patern, grafički korisnički interfejs.*

Abstract: *The research object of this work is the application of software patterns in the creation of a sustainable structure of a software system. Through research, it was necessary to see what places in the program can lead to exponential accumulation of program code and the appearance of chaos when new user requests are added. At these points, the concept of design patterns had to be applied to prevent the creation of an unsustainable and ineffective structure of the software system. The paper explains the concept of design patterns, the advantages of their use in the development of software systems, and the structure of the Abstract Factory sample solution. In the study example, the integration of four heterogeneous Java technologies of the graphical user interface (Swing, JavaFX, SWT and Apache Pivot) was used to show how the concept of patterns can be used to create a sustainable software system architecture. The resulting structure is the basis for efficient upgrading and development.*
Key words: *Software design patterns, Abstract Factory pattern, graphical user interface.*

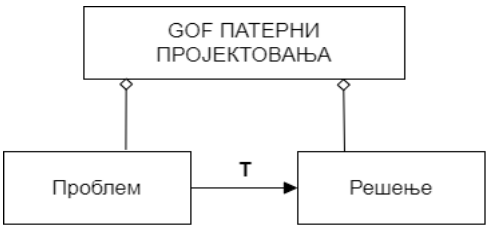
1. UVOD

Usled velike konkurencije i brzog razvoja tehnologija, korisnički zahtevi se veoma često menjaju. Zbog toga, softverski inženjeri teže da razviju održivo, efikasno i ponovo upotrebljivo softversko rešenje. Prilikom razvoja takvih sistema, neophodno je

predvideti koji delovi programa mogu dovesti do složenih i komplikovanih struktura i algoritama koji se eksponencijalno povećavaju sa pojavom novih korisničkih zahteva. Softversko rešenje u kojem se programski kod eksponencijalno povećava je nekvalitetno, neodrživo, neefikasno i ne može se prilagoditi spoljnim uticajima.
Kod softverskih sistema koji se mogu ponovo upotrebiti cilj je prepoznati koji delovi programa su specifični (promenljivi) za domen problema, a koji delovi programskog koda su opšti (nepromenljivi) i mogu se upotrebiti za rešavanje grupe sličnih problema. Prepoznavanje i kreiranje generičkih struktura u softverskom sistemu zahteva vreme, iskustvo i znanje programera i ne može se postići odmah. Jedan od načina da se kreira takav softverski sistem jeste korišćenjem opšte prihvaćenih rešenja skupa sličnih problema, odnosno softverskih paterna.
Softverski inženjeri predstavljaju paterne kao strukturu koju čini ime, problem (pojava koja se ponavlja u određenom kontekstu), rešenje (skup komponenata, njihove odgovornosti i međusobna interakcija) i posledice (pojave koje nastaju primenom rešenja paterna na problem u određenom kontekstu) [1]. Prekretnica u teorijskom i praktičnom proučavanju softverskih paterna predstavlja knjiga *„Design patterns: elements of reusable object-oriented software",* u kojoj je opisano 23 paterna projektovanja. Paterni su podeljeni u tri grupe: kreacioni, strukturni i paterni ponašanja. ***Abstract Factory*** patern, koji je upotrebljen u ovom radu, spada u grupu kreacionih paterna projektovanja.
Rad je organizovan u šest poglavlja. Nakon uvodnog poglavlja, sledi objašnjenje paterna projektovanja (drugo poglavlje) i Abstract Factory paterna (treće poglavlje). U četvrtom poglavlju je objašnjen održivi softverski sistem u kojem je primenjen Abstract Factory paterna. Peto poglavlje sadrži zaključna razmatranja i pravce daljeg istraživanja.

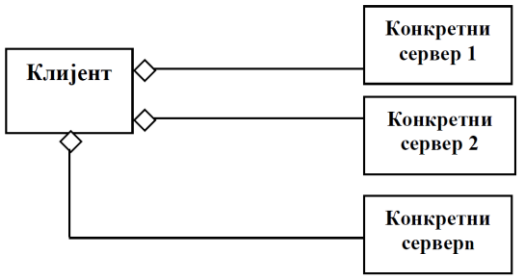
2. PATERNI PROJEKTOVANJA

Paterni projektovanja predstavljaju važan alat u softverskom inženjerstvu za kreiranje održivih, fleksibilnih i efikasnih softverskih struktura. Paterni projektovanja se mogu definisati kao šablon za rešavanje problema u različitim situacijama razvoja softvera. Oni obuhvataju generička rešenja koja se mogu upotrebljavati neograničeni broj puta za rešavanje grupe srodnih problema. Problem paterna objašnjava u kojim situacijama treba primeniti određeni patern. Rešenje paterna opisuje komponente, njihove odgovornosti i način međusobne interakcije. U suštini, paterni obezbeđuju način transformacije problema u rešenje.



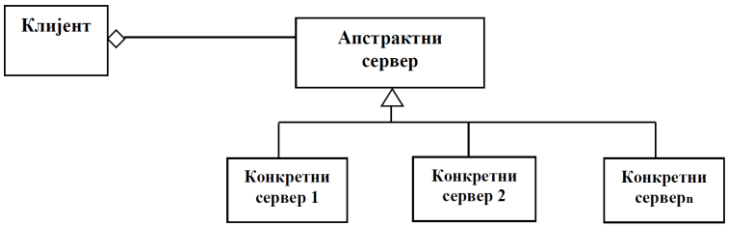
Slika 1: Transformacija strukture problema u strukturu rešenja [2]

Struktura problema paterna sadrži entitet *Klijent* koji izvršava svoju ulogu koristeći funkcionalnosti *Konkretnih servera.* U strukturi problema, *Klijent* direktno zavisi od *Konkretnih servera* i podložan je promenama prilikom dodavanja novih funkcinalnosti sistema (novi *Konkretni serveri*).



Slika 2: Struktura problema paterna projektovanja [3]

Nasuprot strukturi problema paterna, postoji i struktura rešenja paterna projektovanja. Primećeno je da 20 od 23 paterna projektovanja ima istu strukturu rešenja koju čini *Klijent, Apstraktni server* i *Konkretni serveri* (Vlajić, 2014, str. 10). *Klijent* posredstvom *Apstraktnog server* pristupa *Konkretnim serverima* i njihovim funkcionalnostima kako bi izvršio svoju ulogu. *Apstraktni server* obezbeđuje apstraktne funkcionalnosti koje *Konkretni serveri* implementiraju.



Slika 3: Struktura rešenja paterna projektovanja [3]

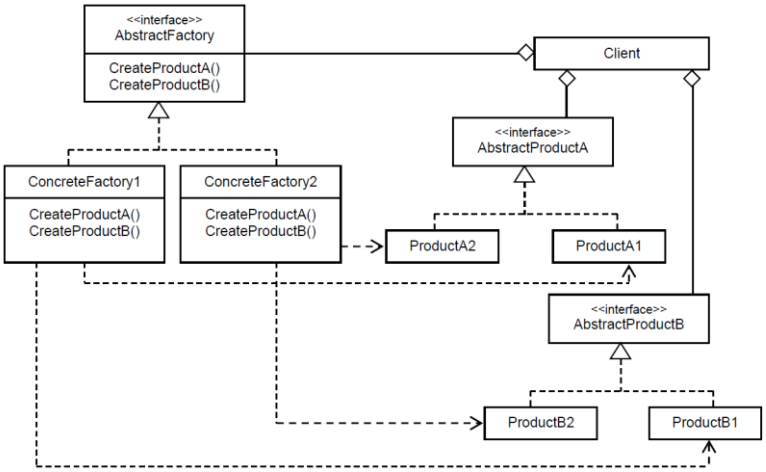
Gore navedena transformacija stukture problema u strukturu rešenja čini softverski sistem održivim, sposobnim da se razvija efikasno i sa malim izmenama usled pojave novih korisničkih zahteva (funkcionalnosti).

3. ABSTRACT FACTORY PATERN

Abstract Factory patern jeste kreacioni patern projektovanja koji definiše interfejs pomoću kojeg se kreira grupa srodnih i zavisnih objekata, bez direktnog navođenja konkretnih implementacija datog interfejsa [4]. Abstract Factory patern onemogućuje klijentu pristup konkretnim proizvodima i udaljava ga od odgovornosti kreiranja istih.

Odgovornost kreiranja proizvoda učaureno je u generičkom interfejsu (*Abstract Factory*). U okviru strukture rešenja Abstract Factory paterna postoje entiteti:

- *Client* koji ima odgovornost da inicira kreiranje nove instance proizvoda i da nadgleda taj proces;
- *AbstractFactory* je interfejs koji definiše operacije za kreiranje grupe sličnih i zavisnih proizvoda;
- *ConcreteFactory* je klasa koja implementira interfejs *Abstract Factory* i njegove operacije za kreiranje proizvoda;
- *AbstractProduct* definiše interfejs grupe srodnih i povezanih proizvoda;
- *ConcreteProduct* predstavlja klasu konkretnog proizvoda (*ProductA1, ProductA2, ProductB1* i *ProductB2*).



Slika 4: Struktura rešenja Abstract Factory paterna [3]

Abstract Facotry patern treba primeniti u sledećim situacijama:

- Kada sistem samostalno odlučuje koji objekat će se kreirati, bez uticaja klijenta. Sistem je nezavisan;
- Kada sistem sadrži više grupa sličnih i zavisnih proizvoda;

Primenom Abstract Factory paterna ostvaruje se bolja kontrola nad procesom kreiranja proizvoda i jednostavnije dodavanje novih implementacija proizvoda.

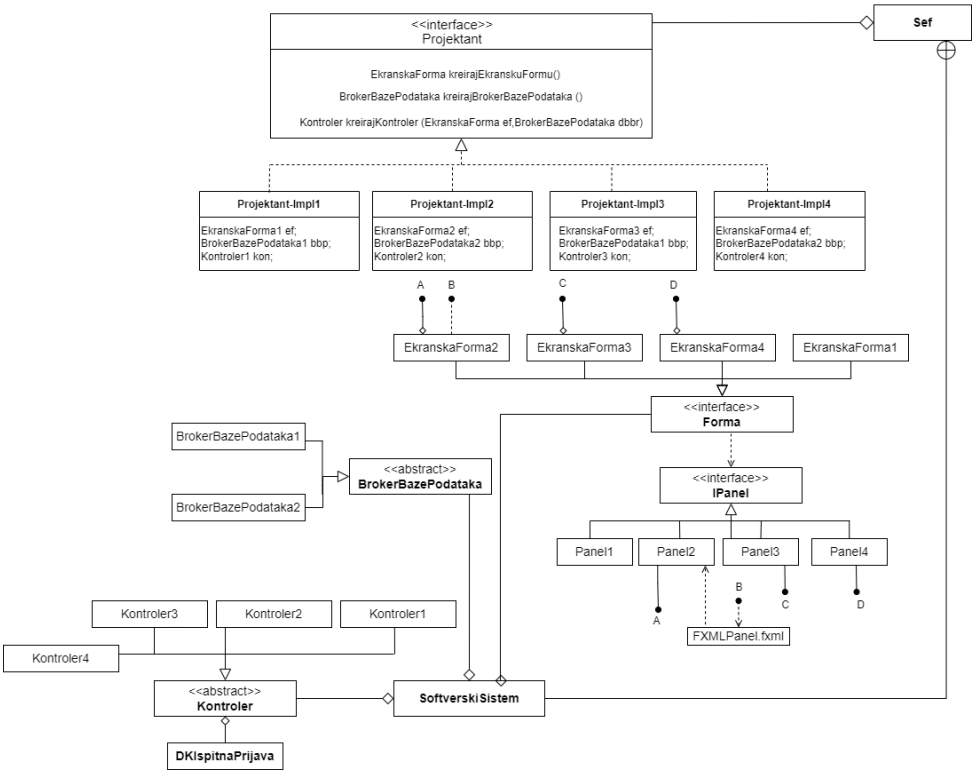
4. STUDIJSKI PRIMER

Studijski primer ovog rada ima za cilj da prikaže strukturu rešenja problema kreiranja složenih sistema. U softverskom rešenju treba da postoji ravnoteža između opštih i specifičnih delova programa, tako da u isto vreme zadovolji trenutni problem i buduće korisničke zahteve.

Održiva softverska struktura koju mi predlažemo, odnosi se na softverski sistem za obradu prijava studenata. Softversko rešenje implementirano je korišćenjem Java programskog jezika i u sebi integriše četiri Javine tehnologije za kreiranje grafičkog

korisničkog interfejsa (Swing, JavaFX, SWT i Apache Pivot). Održivost kreirane strukture primećuje se prilikom dodavanja novih implementacija ekranskih formi, odnosno nove tehnologije grafičkog korisničkog interfejsa. Integracija heterogenih tehnologija tako da čine homogenu strukturu realizovana je korišćenjem kreacionog paterna projektovanja, Abstract Factory paterna.

Softverski sistem za obradu prijava studenata se sastoji od ekranske forme, na kojoj se nalaze grafičke komponente pomoću kojih korisnik komunicira sa sistemom, brokera baze podataka i kontrolera. Broker baze podataka izvršava upite nad skladištem podataka, dok kontroler predstavlja posrednika između ekranske forme i broker baze podataka. Shodno elementima softverskog sistema, strukture rešenja Abstract Factory paterna i korisničkog zahteva da sistem sadrži četiri GUI tehnologije, kreirana je struktura sa Slike 5.

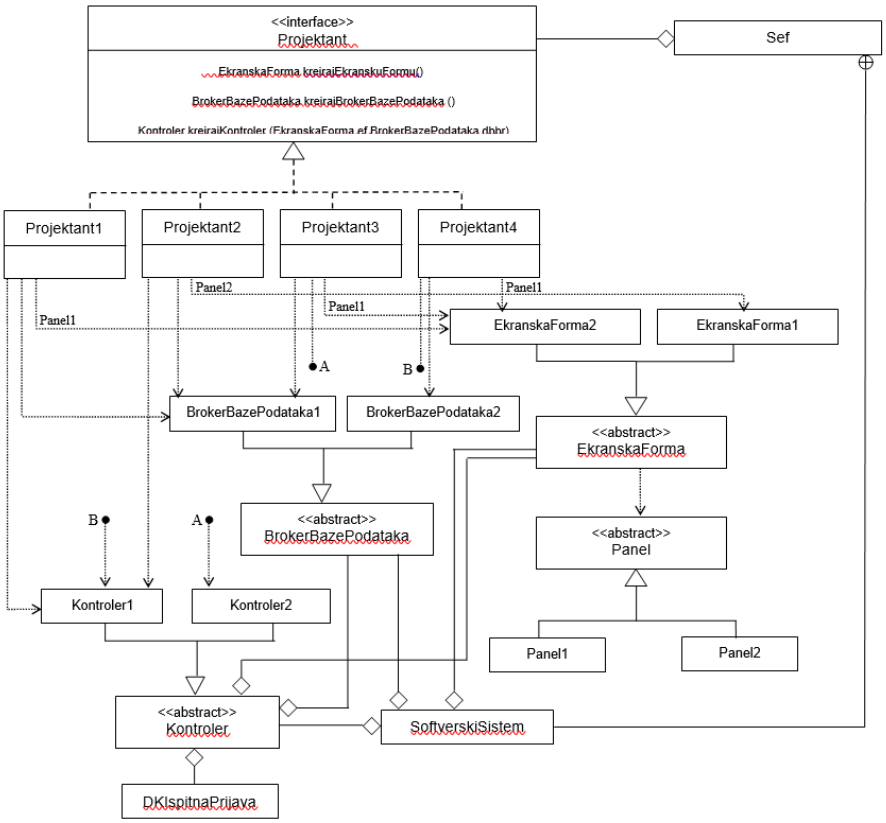


Slika 5: Dijagram klasa softverskog sistema

U kreiranoj strukturi, klasa *Šef* analogna je klasi *Client* iz definicije Abstract Factory paterna (Slika 4.). *Šef* ima odgovornost da inicira i nadzire proces kreiranja složenog proizvoda, odnosno klase *SoftverskiSistem*. Odgovornost kreiranja delova složenog proizvoda (*Forma, Kontroler* i *BrokerBazePodataka*) ima interfejs *Projektant*, odnosno klase koje ga implementiraju (*Projektant-Impl1*, *Projektant-Impl2, Projektant-Impl3* i *Projektant-Impl4*). Interfejs *Projektant* odgovara interfejsu *AbstractFactory* (Slika 4.),

dok su klase *Forma*, *Kontroler* i *BrokerBazePodataka* analogne klasi *AbstractProduct* Abstract Factory paterna. U kreiranoj strukturi, *Šef* nema direktan pristup implementacijama delova složenog proizvoda, već njima pristupa preko interfejsa *Projektant*. *Šef* je spoljni entitet koji ne može da utiče na funkcionalnosti sistema i odlučivanje koji tip proizvoda će se kreirati. Suština ovako dobijene strukture jeste da se postigne nezavisnost sistema i da dodavanje novih implementacija delova složenog proizvoda bude jednostavno (nove implementacije interfejsa *EkranskaForma* i *Kontroler*).

Na početku razvoja softverskog sistema, korisnički interfejs implementiran je korišćenjem Swing tehnologije i strukture rešenja Abstract Factory paterna. Struktura tog sistema prikazana je na Slici 6.



Slika 6: Dijagram klasa - Prvo softversko rešenje

U cilju ispitivanja održivosti, fleksibilnosti i efikasnosti tako kreirane strukture (Slika 6.), javlja se ideja dodavanja novih ekranskih formi i novih tehnologija korisničkog interfejsa. S obzirom da je korisnički interfejs softverskog sistema nezavisan od ostatka sistema (ekranska forma sadrži odgovarajući nivo apstrakcije i ne definiše logiku

reagovanja na događaje, već samo izglede grafičkih komponenata) dodavanje novih implementacija ekranske forme je bilo jednostavno. Dodat je jedan nivo apstrakcije, tj. interfejsi Forma i IPanel (Slika 5.) koji deklarišu metode koje se mogu primeniti na ekranske forme i panele implementirane različitim tehnologijama korisničkog interfejsa. Ostatak sistema nije izmenjen jer su primenom paterna eliminisana mesta u programu koja bi dovodila do nagomilavanja programskog koda i do neodržive strukture.
Softverski sistem sa Slike 6. obogaćen je novim klasama koje sadrže tehnologije JavaFX (*Kontroler2, EkranskaForma2* i *Panel2*), SWT (*Kontroler3, EkranskaForma3* i *Panel3*) i Apache Pivot (*Kontroler4, EkranskaForma4* i *Panel4*), a kao rezultat je dobijena struktura sa Slike 5. Ovime je pokazano da je kreirana struktura održiva i da ne zahteva značajne promene, prilikom dodavanja novog korisničkog zahteva (zahtev da se dodaju nove tehnologije korisničkog interfejsa, tj. nove implementacije ekranskih formi).
Abstract Factory patern omogućava da klijent komunicira sa sistemom posredstvom apstrakcija, pa tako promene u sistemu nisu vidljive spolja (entitetu *Šef*). Takođe, u ovako kreiranoj strukturi moguće je dodavati veliki broj implementacija interfejsa *Projektant, Forma, Kontroler* i *BrokerBazePodataka,* a da ostatak klasa ostanu nepromenjene. Nedostatak koji je prisutan u softverskom sistemu odnosi se na ograničenost kolekcije proizvoda (delova složenog proizvoda) koje se mogu kreirati. To podrazumeva da svaka nova ekranska forma (panel) koja se kreira mora imati iste grafičke komponente koje su definisane interfejsom *Forma* (*IPanel*). Ovaj nedostatak je prisutan i kod funkcionalnosti kontrolera i brokera baze podataka.

5. ZAKLJUČAK

Da bi softverski sistem bio održiv neophodno je da njegova osnova bude stabilna i nepromenljiva. Struktura softverskog sistema treba da bude takva da se može jednostavno prilagoditi novim funkcionalnostima i/ili novim tehnologijama uz minimalne izmene. Jedan od načina da se kreira konzistentna i prilagodljiva struktura softverskog sistema jeste primenom softverskih paterna. U našem studijskom primeru pokazali smo da primenom kreacionog paterna projektovanja, Abstract Factory paterna, može se kreirati homogena i skladna struktura. Kreirana struktura ostaje nepromenjena sa dodavanjem novih tehnologija korisničkog interfejsa.
Karakteristike softverskog sistema koje ga čine održivim jesu: entiteti komuniciraju preko interfejsa, klijent sadrži reference na apstrakcije i nema direktan pristup delovima sistema (ne može ih izmeniti), ekranske forme su nezavisne od drugih delova sistema.
Pravci daljeg istraživanja odnose se na ispitivanju otpornosti i održivosti kreiranog softverskog sistema, prilikom dodavanja novih korisničkih zahteva. Cilj je rešiti nedostatke koji su uočeni i dobiti softverski sistem koji će u sebi integrisati različite *desktop, android* i *web* tehnologije korisničkog interfejsa.

LITERATURA

[1] Alexander, C., Alexander, P. I. T. D. O. a. C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Shlomo, A. (1977). A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

[2] Vlajić, S., Vojislav, S., Savic, D., & Lazarević, S. D. (2016). The General Form of GoF Design Patterns. ResearchGate.
[3] Vlajić, S. (2014). Softverski Paterni. Zlatni Presek.
[4] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH.