Komponentno programiranje

Jelena Šimpraga
jelence17@gmail.com

28. oktobar 2018.

1 Uvod

U tekstu koji sledi ukratko su prikazani osnovni principi *komponentnog* (eng. *component programming*) programiranja.

Pod komponentno-objektnim programiranjem podrazumeva se princip raslojavanja problema na sekcije, odnosno komponente, od kojih svaka ima specijalnu ulogu ili domen u rešavanju problema.

U odeljku 2 precizirane su osnovne osobine, dok su razlike u odnosu na objektno-orijentisano programiranje date u odeljku 3.

2 Osobine komponentnog programiranja

Komponentno programiranje predstavlja vid organizovanja koda u slabo spregnute celine, koje predstavljaju zasebne entitete i obavljaju određene celine posla. Posebno je pogodno za ponovno iskorišćavanje napravljenih komponenti, pridruživanje i spajanje komponenti u veće celine ili kao obične kontejnere. Kada se kreira novi entitet, funkcionalnosti se entitetu dodaju preko komponenti koje su često napravljene univerzalno. Te komponente mogu biti elementi grafike, skriptovi, simulacija fizike, zvukovi, 3D modeli i još mnogo toga. Entiteti sami za sebe nisu ništa drugo do prosti kontejneri, u kojima su sadržane sve neophodne komponente za funkcionisanje određenog entiteta.

Što se tiče sistemske koordinacije, komponente komuniciraju jedne sa drugima preko interfejsa [1]. Kada komponenta pruža usluge ostatku sistema, ona ih usvaja pod uslovom da interfejs navodi usluge koje druge komponente mogu da koriste i kako one mogu da rade. Ovaj interfejs se može posmatrati kao potpis komponente - klijent ne treba da zna o unutrašnjim poslovima komponente (implementacija) kako bi ih koristio. Ovaj princip rezultata u komponenti naziva se enkapsulacija [2].

Još jedna važna osobina komponenti je da su zamenljive, tako da komponenta može da zameni drugu, ako je komponenta naslednik i ispunjava zahteve početne komponente (izražene preko interfejsa). Prema tome, komponente se mogu zameniti bilo sa ažuriranom verzijom ili alternativom bez narušavanja sistema u kojoj komponenta deluje.

Ponovna upotrebljivost je važna karakteristika kvalitetne komponente. Programeri treba da osmisle i sprovedu komponente na takav način da mnogi različiti programi mogu ponovo da ih koriste.

Model komponenti predstavlja definisanje standarda za implementaciju, dokumentaciju i primenu komponenti. Primeri komponentnih modela su: Enterprise JavaBeans (EJB), Component Object Model (COM), NET model i Corba model. Komponentni model određuje način na koji interfejsi treba da budu definisani i koji elementi treba da budu uključeni u definiciju interfejsa.

3 Razlike u odnosu na OOP

Fundamentalna razlika između ove dve metodologije leži u njihovom različitom shvatanju konačne aplikacije. U tradicionalnom objektno-orijentisanom svetu, iako se pravi faktorizacija problema na klase, jednom kada se one iskompajliraju, rezultat predstavlja monolitni, tj. jedinstveni binarni kod. Sve klase dele istu jedinicu za fizicko raspoređivanje, proces, adresni prostor, privilegije bezbednosti itd. Ako više programera radi na istom baznom kodu oni moraju da dele izvorne datoteke. U takvoj aplikaciji promena u jednoj klasi može izazvati masovno ponovno povezivanje cele aplikacije i zahteva ponovno testiranje i preraspoređivanje svih drugih klasa.

S druge strane, komponentno-orijentisana aplikacija sadrzi kolekciju interakcije binarnih aplikativnih modula tj. njihovih komponenti i poziva koji ih vezuju.

Zasebno, binarna komponenta ne može mnogo da uradi sama. Neke komponente mogu biti za generalnu namenu, dok druge mogu biti specijalizovane i razvijene posebno za datu aplikiaciju. Aplikacija implementira i izvršava svoju logiku spajanjem funkcionalnosti koje nude pojedinačne komponente. Tehnologije koje omogućavaju komponentno programiranje, kao sto su COM, J2EE, CORBA, i .NET [1] obezbeđuju "vodovod" (eng. plumbing), tj. neophodne infrastrukture koje omogućavaju povezivanje komponenti na neprimetan način, dok je glavna razlika izmedju ovih tehnologija lakoća sa kojom se ostvaruje povezivanje komponenti.

Motivacija za razbijanje monolitne aplikacije na više binarnih komponenti analogna je postavljanju koda za različite klase u različite fajlove. Postavljanjem koda za svaku klasu u sopstveni fajl oslobađaju se sprege među klasama i programeri koji su odgovorni za njih. Ako se naprave promene u jednoj klasi, iako je neophodno ponovo povezati celu aplikaciju, samo je potrebno rekompajlirati izvorni kod za datu klasu.

Ukratko, objektno-orijentisano programiranje se fokusira na odnose među klasama koje se ujedinjuju u jednu veliku binarnu izvršnu celinu, dok se komponentno-orijentisano programirnje fokusira na izmenjive module koda koji rade nezavisno, pri čemu se za njihovu upotrebu ne zahteva poznavanje njihove unutrašnjosti.

Objektno orijentisana paradigma zasnovana je na modelovanju objekata nalik na ono što predstavljaju u realnom svetu, tj. da se učini lakšim za čitanje ljudima (što programerima to i krajnjim korisnicima).

Sa druge strane, komponentno orijentisani razvoj softvera ne predlaže ništa slično i umesto toga savetuje da bi programeri trebalo da konstruišu softver spajanjem ranije napravljenih komponenti - slično kao u elektrotehnici ili mehanici. Ovo je takođe poželjno i zbog lakog održavanja i prepravljanja komponenti, a ponekad i kompletnom zamenom drugom komponentom.

Lakše je proširiti komponentno-orijentisanu aplikaciju jer se novi zahtevi za

sprovođenje mogu obezbediti u novim komponentama, bez diranja postojećih komponenti na koje ti zahtevi ne utiču. Ovi faktori omogućavaju smanjenje troškova dugoročnog održavanja, faktor koji je bitan u svakom poslu i omogućava objašnjenje za rasprostranjeno usvajanje komponentnog programiranja. Na primer, razmotrimo brz razvoj koji uživaju mnogi Visual Basic [4] projekti koji se oslanjaju na biblioteke ActiveX kontrola u skoro svakom aspektu aplikacije.

Ipak, komponentno orijentisani razvoj ne isključuje korišćenje objektno orijentisanih principa. Zasebni skriptovi mogu u okviru svoje enkapsulacije imati hijerarhiju klasa, interfejse i sve druge prednosti objektno-orijentisanog jezika. Najbolje stvari ipak mogu se dobiti kombinacijom ova dva principa, što se najčešće i radi.

JDBC database drivers [3] predstavljaju dobar primer komponentno orijentisanog razmišljanja (implementiranog u objektno-orijentisanom svetu).

Literatura

[1] George T. Heineman and William T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[2] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[3] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, 2nd edition edition, 2002.

[4] Peter Wright. *Beginning Visual Basic 6*. Wrox Press Ltd., Birmingham, UK, UK, 1998.