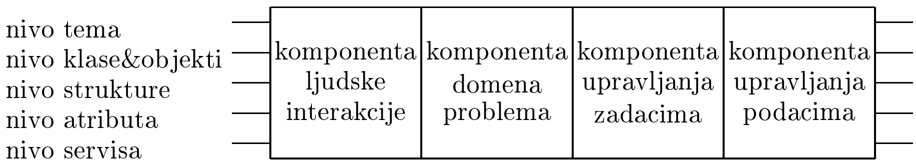
1 Objektno-orijentisano projektovanje (OOP)

Prelaz od objektno-orijentisane analize (OOA) ka OOP predstavlja progresivno širenje istog modela. OOA nivoi modeliraju domen problema i odgovornosti sistema. OOP proširenje OOA nivoa modelira specifičnu implementaciju. Proširenje se postiže dodavanjem komponenti – ljudske interakcije, upravljanja zadacima i upravljanja podacima.

OOP identifikuje i definiše dodatne klase-i-objekte koji odslikavaju implementaciju zahteva (nivo dijaloga, nivo upravljanja zadacima, nivo upravljanja podacima).

OOA i OOP se mogu izvoditi kao strogo sekvencijalne faze, jedna za drugom, ili se mogu preklapati, što je češći slučaj u malim timovima i okruženjima pogodnim za razvoj (npr. Smalltalk).

Višenivoski, višekomponentni model sastoji se od pet nivoa proširenih u fazi projektovanja korišćenjem četiri glavne komponente: ljudske interakcije, domena problema, upravljanja zadacima i upravljanja podacima (slika 1).



Slika 1: Višenivoski, višekomponentni model

Komponenta ljudske interakcije uključuje aktuelne displeje i ulaze potrebne za efikasnu interakciju, na primer klase uključuju Prozor i Selektor.

Unutar komponente domena problema zamenjuju se neki rezultati OOA zbog moguće potrebe za dekompozicijom ili spajanjem nekih klasa&objekata, struktura, atributa, servisa, dobijenih u OOA. Mogući razlozi za ovakve izmene su nagodbe oko rokova i dimenzija projekta, podela zbog memorijskih ograničenja, podela zbog raspodele na pojedine članove tima, ili prilagodjavanje klasama specifičnim za domen koje su već na raspolaganju za ponovnu upotrebu.

Komponenta upravljanja zadacima uključuje definicije zadataka koje treba izvršiti u realnom vremenu, komunikaciju i koordinaciju. Ona uključuje i razmatranje alokacije hardvera i protokola spoljašnjih sistema i uredjaja. Primeri klasa u ovoj komponenti su UpravljačZadacima, Zadatak.

Komponenta upravljanja podacima uključuje pristup i upravljanje perzistentnim podacima. Primeri klasa ove komponente su Relacija, PrimarniKljuč, DeljiviKatanac, EkskluzivniKatanac, Autorizacija, Transakcija (za upravljač relacionom bazom), itd.

Dok je OOA nezavisna od programskog jezika, preliminarno OOP je takodje velikim delom nezavisno od programskog jezika, ali je detaljno OOP zavisno od jezika i može biti implementirano na proceduralnim, paketno-orijentisanim i objektno-orijentisanim programskim jezicima.

Notacija i pristup OOP izgradjeni su na tri stalno korišćene metode organizacije koje opredeljuju ljudsko mišljenje u razumevanju realnog sveta:
– deljenje iskustva u posebne objekte i njihove atribute (npr. drvo/veličina/prostorni odnos prema drugim objektima);
– razlikovanje celine i njenih delova (npr. drvo/grane);
– formiranje raznih klasa objekata i razlikovanje tih klasa (npr. drveće/kamenje).

1.1 Osnovna terminologija

U OOP, objekat (ili instanca) jeste apstrakcija nečega iz domena problema ili njegove implementacije, o čemu sistem ima mogućnost čuvanja informacije, ili sa čime sistem može da interaguje. To je učaurenje vrednosti atributa i njihovih servisa.

Klasa je opis jednog ili više objekata koji se mogu opisati istim skupom atributa i servisa; pored toga, klasa opisuje i kako se kreira novi objekat te klase.

Klasa&Objekti označava klasu i objekte te klase.

Analiza je proces proučavanja domena problema koji vodi ka specifikaciji ponašanja; kompletan, konzistentan i moguć opis onoga što je potrebno; funkcionalne i kvantifikovane karakteristike (npr. pouzdanost, raspoloživost, performanse).

Projektovanje je dopuna specifikacije ponašanja detaljima koji su potrebni za stvarnu računarsku implementaciju, uključujući detalje ljudske interakcije, upravljanja zadacima i upravljanja podacima. Dakle, projektant ima zadatak da pretoči rezultate analize (npr. specifikaciju) u specifičnu hardversko/softversku implementaciju.

1.2 Izrada prototipova

Razlozi za izradu prototipa, koja se često primenjuje u fazi projektovanja, jesu:
– dinamičnost
– mogućnost eksperimentisanja sa projektovanjem komponente ljudske interakcije
– mogućnost otkrivanja propuštenih zahteva
– testiranje projekta
– postizanje funkcionalnosti što je pre moguće.

1.3 Pristupi ciklusu razvoja

Projektantski tim može pristupiti ciklusu razvoja na jedan od sledećih načina:
– sekvencijalni: analiza, projektovanje, implementacija;
– spiralni: analiza, prototip; projektovanje, prototip; programiranje, prototip;
– postupni: malo analize, malo projektovanja, malo programiranja; ponoviti;

1.4 OOP pojedinačnih komponenti

Projektovanje komponente **domena problema** polazi od rezultata OOA, poboljšava ih i dodaje im detalje.

Potreba za poboljšanjem rezultata OOA može biti izazvana promenama u zahtevima klijenta, tržišta, ili nedostatkom razumevanja domena problema od strane sistem analitičara ili eksperta iz domena problema. Rezultati OOA jednostavno se menjaju.

Rezultati OOA se u fazi OOP dopunjuju detaljima koji su posledica sledećih dodatnih kriterijuma:

– mogućnost ponovne upotrebe (engl. reusability) klasa iz OOP faze i faze programiranja;
– grupisanje (zajedno) klasa specifičnih za domen problema;
– uspostavljanje protokola dodavanjem klase generalizacije;
– prilagodjavanje nivoa nasledjivanja;
– poboljšanje performansi;
– podrška komponenti upravljanja podacima;
– dodavanje komponenti niskog nivoa;
– ne modifikovati rezultate prema zaduženjima članova tima;

Na primer, pokazaće se da mnoge klase specijalizacije imaju potrebu za istim protokolom. Tada se može uvesti nova klasa koja uspostavlja taj protokol (sa zajedničkim atributima i servisima), pri čemu se detalji servisa mogu definisati u klasama – specijalizacijama.

U OOP **komponente ljudske interakcje**, identifikuju se i dodaju rezultatima analize detalji interakcije, kao što je projektovanje formata prozora i izveštaja. Izrada prototipa pomaže u izboru pravih mehanizama interakcije.

Strategija projektovanja ove komponente uključuje sledeće elemente:
– klasifikacija ljudi koji interaguju sa sistemom, u različite kategorije;
– opis ljudi i scenarija njihovih zadataka;
– projektovanje hijerarhije komandi;
– projektovanje detaljne interakcije;
– nastavak izrade prototipa;
– projektovanje klasa komponente ljudske interakcije;
– projektovanje, uzimajući u obzir grafičke interfejse, kada su na raspolaganju.

Klasifikacija ljudi može početi od generalizacija/specijalizacija strukture "Osoba", ako takva postoji u komponenti domena problema. Na primer, za registraciju vozila to može biti kategorizacija Osoba / Službenik / Vlasnik / Vlasnik-Službenik. Zatim se mogu razmotriti dodatni skupovi ljudi koji će interagovati sa sistemom koji se razmatra. Klasifikacija može biti prema:
– nivou iskustva (početnik / povremeni korisnik / srednji / napredni)
– organizacionom nivou (direktor / šef / osoblje / rukovodilac / službenik)
– pripadnosti različitim grupama (osoblje/potrošač; nastavnik/student)

Projektovanje hijerarhije komandi uključuje:

– proučavanje postojećih mogućnosti korisničke interakcije
– uspostavljanje početne hijerarhije komandi
– doterati hijerahiju komandi.

Hijerarhija komandi može biti predstavljena na više načina:
– kao niz meni-ekrana
– kao meni linija
– kao niz ikona.

Hijerarhija komandi je prezentacija postojećih servisa organizovana korišćenjem proceduralne apstrakcije. Na primer, može se krenuti od početne meni-linije koja sadrži komande:

File Edit Format Calculate Monitor Window

Da bi se hijerarhija komandi doterala, potrebno je razmotriti **uredjenje** (najfrekventniji servisi prvo, i to u redosledu koji nalažu koraci posla korisnika), **deobu celina/deo** (unutar samih servisa)i **deobu po širini i dubini** uzimajući u obzir "kratku memoriju" čoveka koji može da pamti od 5 do 9 elemenata u jednom trenutku. Na primer, početna meni-linija može biti razdeljena u sledeće opcije:

File	Drawing	Edit
New Drawing	Select Filter	Undo
Open Drawing	Document	Cut
Open Item	Overview	Copy
Close	Subject Layer	Paste
Sawe Drawing	Structure Layer	Clear
Sawe Drawing as	Attribute Layer	Select All
Page Setup	Service Layer	Find ...
Print ...		Change
Quit		
Create	Window	Fonts
...	...	...

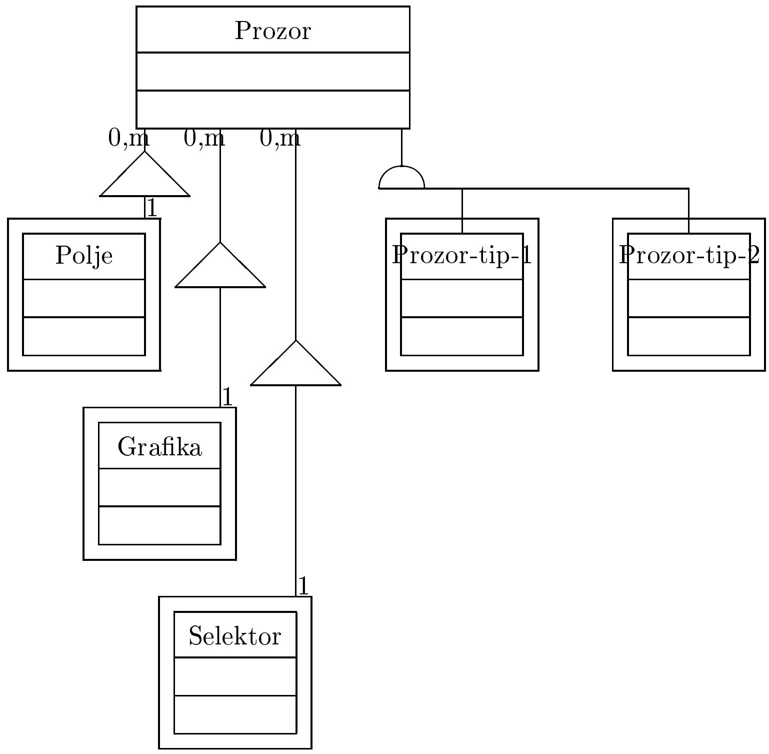
1.5 Projektovanje klasa komponente ljudske interakcije

Ove klase zavise delimično od grafičkog interfejsa koji će se koristiti. Sa projektovanjem ovih klasa može se početi od organizacije ljudske interakcije pomoću prozora i njihovih komponenti (slika 2).

1.6 Projektovanje komponente upravljanja zadacima

Zadatak (engl. task) je drugo ime za proces (struja aktivnosti definisana svojim programskim kodom). Konkurentno izvršenje većeg broja zadataka zove se multitasking.

Posebni zadaci izdvajaju ponašanja koja se mogu odvijati istovremeno. To konkurentno ponašanje se može realizovati na različitim procesorima a može se i simulirati korišćenjem jednog procesora u sprezi sa multitasking operativnim sistemom. Alternativa za multitasking je ciklično izvršenje sekvencijalnog programa.



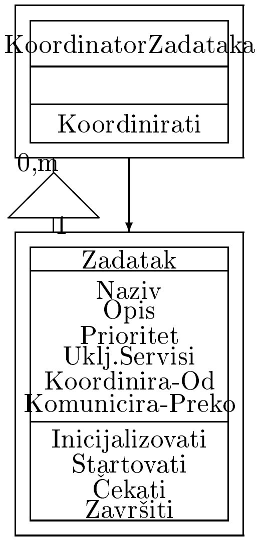
Slika 2: Prozori i komponente

Izvršavaju se mali, brzi delovi programa, posle svakog se proverava šta se dogodilo u medjuvremenu i odgovara se adekvatno.

Izbor zadataka vrši se prema sledećoj strategiji:
– identifikovati zadatke vodjene dogadjajem (engl. event-driven tasks);
– identifikovati zadatke vodjene vremenom (engl. time-driven tasks);
– identifikovati prioritetne zadatke i kritične zadatke;
– identifikovati koordinatora;
– razmotriti svaki zadatak;
– definisati svaki zadatak.

Zadaci vodjeni dogadjajem mogu biti odgovorni za, na primer, komunikaciju sa uredjajem, jednim ili većim brojem prozora na ekranu, itd. Zadaci vodjeni vremenom se obično koriste za periodičnu kominikaciju interfejsa, podsistema, procesora, zadataka ili drugih sistema.

Broj zadataka treba da je što manji. Potreba za zadatkom–koordinatorom javlja se kod postojanja tri ili više drugih zadataka. Njegovo ponašanje se može opisati matricom prelazaka stanja, i predstavlja učaurenje medjuprocesne koordinacije. Dijagramski se komponenta upravljanja zadacima može prikazati slikom 3.



Slika 3: OOP notacija za komponentu upravljanja zadacima

Svaki zadatak na slici 3 definisan je imenom, opisom, prioritetom, servisima, oznakom koordinatora i načinom komunikacije.

1.7 Komponenta upravljanja podacima

Komponenta upravljanja podacima obezbedjuje infrastrukturu za skladištenje i pretraživanje objekata iz sistema za upravljanje podacima.

Ova komponenta izoluje uticaj modela podataka, bilo da je to datotečni, relacioni, objektno-orijentisani ili neki drugi model.

Projektovanje ove komponente treba da uključi i projektovanje **reprezentacije podataka** i projektovanje odgovarajućih **servisa**.

Za datotečni model, projektovanje reprezentacije podataka se sastoji iz sledećih koraka:

– definisati tabele u 1NF, i to tako što se navedu sve klase i njihovi atributi, a zatim se ta lista prevodi u definicije 1NF tabela;

– definisati datoteku za svaku 1NF tabelu;

– proceniti performanse i potrebe za prostorom na disku;

– odustati, po potrebi, od 1NF da bi se zadovoljili zahtevi performansi i prostora; ako je potrebno, grupisati atribute cele generalizacija/specijalizacija strukture u jednu datoteku; time se smanjuje broj datoteka (na račun izvesnog praznog prostora za prazna polja); ako je potrebno, kombinovati neke vrednosti atributa u jednu kodiranu vrednost, umesto većeg broja polja; time se smanjuje količina potrebnog prostora (na račun potrebe za dodatnom obradom).

Za relacioni sistem za upravljanje bazama podataka, koraci su sledeći:

– definisati tabele u 3NF, i to tako što se navedu sve klase i njihovi atributi, a zatim se ta lista prevodi u 3NF definicije tabela;

– definisati tabelu baze podataka za svaku tabelu iz prethodnog koraka;

– proceniti performanse i potrebe za prostorom na disku; zatim odustati, po potrebi, od 3NF da bi se zadovoljili zahtevi performansi i prostora.

Za objektno-orijentisani sistem za upravljanje bazama podataka, koraci su sledeći:

– za prošireni relacioni pristup, primeniti isti postupak kao kod relacionih sistema;

– za pristup perzistentnom proširenju OO programskih jezika, može se ispostaviti da nije potrebno ništa uraditi da bi se atributi normalizovali, jer DBMS sam preslikava vrednosti objekata u perzistentnu memoriju.

Pri projektovanju servisa, dodaje se po jedan (implicitni) atribut i jedan (implicitni) servis svakoj klasa&objekti koja ima objekat koji se trajno pamti. Taj atribut tj. servis, "store myself", opisuje način fizičkog predstavljanja objekta te klase. Na primer, kod relacionih sistema, objekat mora da zna kojoj tabeli da pristupi, kako da pristupi potrebnim vrstama, kako da pretraži stare vrednosti (ako postoje), i kako da ih ažurira novim. Potrebno je definisati dodatnu klasu sa objektima, ObjekatServerKlasa&Objekti, sa servisima koji (a) govore svakom objektu da zapamti sebe, (b) pretražuju zapamćene objekte (traženje, uzimanje vrednosti, kreiranje i inicijalizacija objekata) da bi ih koristile druge komponente projektovanog sistema.

1.8 OOP i OO programski jezici

OO projektovanje moguće je implementirati različitim OO programskim jezicima (OOPJ) (a i onima koji to nisu). Pri izboru OOPJ, uzima se u obzir podrška klasama, objektima, generalizaciji/specijalizaciji, strukturi celina/deo, atributima i servisima. Obično se u obzir uzimaju OO jezici C++, ObjectPascal, Smalltalk, Objective-C, Eiffel, Java, ili proceduralni jezici (ne OO) Ada, C, itd. OO jezici su uglavnom algortamski i strukturni jezici. Oni obezbedjuju bar koncepte klase, objekta, nasledjivanja, dinamičkog povezivanja i učurenja (enkapsulacije).

Sledeća tabela sumira karakteristike glavnih OO jezika:

	Ada 95	C++	Eiffel	Java	Smalltalk
Paketi	da	ne	ne	da	ne
Nasledjivanje	jednostruko	višestruko	višestruko	jednostruko	jednostruko
Tipiziranost	stroga	stroga	stroga	stroga	netipiziran
Povezivanje	dinamičko	dinamičko	dinamičko	dinamičko	dinamičko
Paralelizam	da	ne	ne	da	ne
Sakupljanje otpadaka	ne	ne	da	da	da
Predikati	ne	ne	da	ne	ne
Perzistentnost	ne	ne	ne	ne	ne

Jedno od merila dobrog projekta je što manja količina kopčanja (engl. coupling), tj. veza izmedju klasa i veza izmedju objekata. Ako promene na jednoj klasi izazovu lančane promene na nizu drugih klasa, projekat sadrži previše kopčanja i treba ga ponovo razmotriti.

Veza poruka (engl. message connection) treba da je što jednostavnija. Ako sadrži više od tri parametra, razmotriti mogućnost njenog pojednostavljenja.