Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicații şi Tehnologia Informației

Programare orientata pe obiect din perspectiva ingineriei software :

Dezvoltarea orientata pe obiect a aplicatiilor

Mihai Ignat
Grupa 442A

Cuprins:

Introducere

Un sistem obiect-orientat este alcătuit din obiecte care interactionează si care mentin propria lor stare locala si oferă operatiuni in acea stare (Figura 1). Reprezentarea starii este privata si nu poate fi accesata direct din afara obiectului. Procesul de design obiect-orientat implică proiectarea claselor de obiecte si a relatiilor dintre aceste clase. Aceste clase definesc obiectele din sistem si interactiunile lor. Când proiectul este realizat ca un program de executare, obiectele sunt create dinamic pornind de la aceste definitii de clasă.

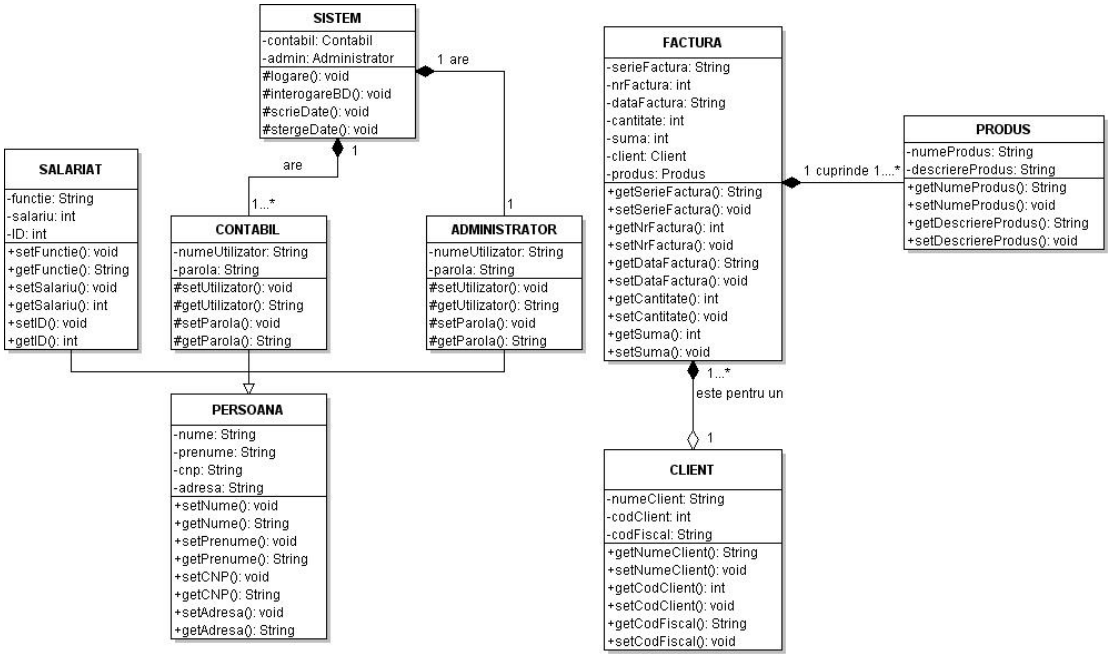

Figura 1 : Exemplu de diagrame de clasa UML

Termenii obiect si obiect-orientat sunt folosite pentru diferite tipuri de entităti, metode de design, sisteme si limbaje de programare. Există o acceptare generală că un obiect este o încapsulare de informatii, iar acest lucru se reflectă în definitia urmatoare a unui obiect si o clasă de obiecte:

Un obiect este o entitate care are o stare si un set definit de operatii care functioneaza in acea stare. Starea este reprezentata ca un set de atribute de obiecte. Operatiile asociate cu obiectul ofera servicii catre alte obiecte (clienti), care solicita aceste servicii atunci când unele operatii calcul sunt necesare. Obiectele sunt create în functie de definitia clasei. O definitie de clasa de obiecte este atât o specificatie de tip cat si un sablon pentru crearea de obiecte. Aceasta include declaratii ale tuturor atributelor si operatiile care ar trebui să fie asociate cu un obiect din acea clasă.

Pasii dezvoltarii obiect-orientate a aplicatiilor

Dezvoltarea aplicatiilor obiect-orientate urmareste urmatorul principiu: fiecare modul din sistem denota un obiect sau o clasa de obiecte din spatiul problemei.

Abstractizarea si incapsularea sunt fundamentale in dezvoltarea obiect-orientata a aplicatiilor. O abstractizare reprezinta o descriere simplificata sau o specificatie a sistemului care retine cateva din detaliile sau proprietatile sistemului in timp ce restul proprietatilor sunt ignorate. Incapsularea sugereaza descopunerea sistemului bazata pe principiul ascunderii deciziilor de design asupra abstractizarilor.

Abstractizarea si incapsularea fac parte din activitatile naturale. Omul tinde sa abstractizeze zilnic prin crearea de modele ale realitatii prin identificarea obiectelor si operatiilor care exista la fiecare nivel al interactiunii. Deci, de exemplu in cazul conducerii unui autoturism, se pot considera modele accereratia, vitezele, volanul si frana (printre altele) si operatiile care pot fi efectuate cu acestea dar si rezultatele lor. Un alt exemplu este repararea unui motor, unde se pot considera obiecte la un nivel scazut de abstractizare pompa de combustibil, carburatorul si distributia.

Similar, un program care implementeaza un model al realitatii (precum ar trebui toate programele sa implementeze) poate fi privit ca un set de obiecte care interactioneaza intre ele. Design-ul obiect-orientat este o parte a dezvoltarii obiect-orientate a programelor în care este folosită o strategie obiect-orientata de-a lungul procesului de dezvoltare. Un proces de dezvoltare a unei aplicatii obiect-orientate are urmatorii pasi:

- Analiza obiect-orientata se ocupă cu dezvoltarea unui model obiect-orientat a domeniului de aplicare. Obiectele din acel model reflecta entitătile si operatiile asociate cu problema care trebuie rezolvată:

    1. Identificarea obiectelor si a atributelor acestora.
    2. Identificarea operatiilor pe care obiectele le pot efectua si care sunt necesare fiecarui obiect

- Design-ul obiect-orientat se ocupă cu dezvoltarea unui model obiect-orientat a unui sistem software pentru a implementa cerintele identificate. Obiectele dintr-un design obiect-orientat sunt legate de rezolvarea problemei. Se poate sa existe relatii strânse între unele obiecte ale problemei si unele obiecte ale solutiei , dar în mod inevitabil, proiectantul trebuie să adauge obiecte noi si sa transforme obiecte problemă pentru a implementa solutia:

    3. Stabilirea vizibilitatii fiecarui obiect in relatie cu celelalte obiecte.
    4. Stabilirea interfetei fiecarui obiect.

- Programarea obiect-orientata se referă la realizarea unui design software utilizând un limbaj de programare obiect-orientat, cum ar fi Java. Un limbaj de programare obiect-orientat ofera mijloace pentru a defini clase de obiecte si un sistem run-time pentru a crea obiecte din aceste clase:

    5. Implementarea fiecarui obiect.

Primul pas implica recunoasterea actorilor principali, agentilor si clientilor din spatiul problemei si a rolului lor in modelul realitatii. De exemplu, intr-un sistem de control al calatoriei unei masini, se pot identifica obiectele concrete precum acceleratia, ambreiajul si motorul si obiectele abstracte precum viteza.

Obiectele identificate in acest pas deriva din substantive pe care le folosim pentru descrierea spatiului problemei. Se pot gasi de asemenea obiecte de interes care sunt similare. In aceste situatii, se poate stabili o clasa de obiecte care are mai multe instante. De exemplu, intr-o interfata de utilizator cu ferestre multiple, se pot identifica ferestrele distincte (ferestre de ajutor, ferestre de mesaje, ferestre de comanda) care au caracteristici similare. Fiecare fereastra poate fi considerata o instanta a unei clase de fereastra.

In pasul urmator, de identificare a operatiilor, este caracterizat comportamentul fiecarui obiect sau fiecarei clase de obiecte. In cadrul acestui pas, se stabilesc semanticele statice ale obiectelor prin determinarea operatiilor care pot fi efectuate cu sens pe un obiect sau de catre obiect. Se stabileste, de asemenea, si comportamentul dinamic al fiecarui obiect prin identificarea constrangerilor in timp si spatiu care trebuie sa fie observate. De exemplu, se poate specifica o operatie necesara la un anumit moment.

In cazul unui sistem cu ferestre multiple, ar trebui permise operatiile de deschidere, inchidere, mutare si rescalare a obiectului fereastra. Pentru efectuarea operatiei, ar trebui sa existe constrangerea ca fereastra sa fie deschisa inainte. Similar, se poate crea constrangerea de dimensiune maxima sau minima a ferestrei.

In mod evident, operatiile suferite de un obiect definesc activitatea obiectului in relatie cu celelalte obiecte. Prin identificarea operatiilor, se pot decupla obiectele unul de celalalt. De exemplu, intr-un sistem multi-fereastra, se poate asuma existenta unui obiect terminal si ar putea fi necesara o operatie de mutare de cursor si inserare. Rezultatul este ca se pot deriva obiecte care sunt reutilizabile prin mostenire deoarece acestea nu sunt dependente de alte obiecte specifice, dar sunt dependente de alte clase de obiecte.

Al treilea pas, de stabilire a vizibilitatii fiecarui obiect in relatie cu celelalte obiecte, identifica dependentele statice dintre obiecte si clase de obiecte (ceea ce vad obiectele si cum sunt vazute de alt obiect anume). Scopul acestui pas este de a captura topologia obiectelor din modelul realitatii.

In urmatorul pas, de stabilire a interfetei fiecarui obiect, este produsa o specificatie de modul, folosind notatii corespunzatoare. Astfel, este capturata semantica statica a fiecarui obiect sau a fiecarei clase de obiecte care a fost

stabilita in pasul precedent. Aceasta specificatie ajuta la crearea legatura dintre clientii unui obiect si obiectul in sine. Interfata formeaza legatura dintre exteriorul si interiorul unui obiect.

In ultimul pas, cel de implementare a fiecarui obiect, se alege o reprezentare corespunzatoare pentru fiecare obiect sau clasa de obiecte si se implementeaza interfata specificata in pasul precedent. Acest pas poate implica compunerea sau descompunerea obiectului. Ocazional, va fi gasit un obiect care este compus din mai multe obiecte subordonate si in acest caz, se repeta metoda pentru descompunerea obiectului. Mai des, un obiect va fi implementat prin descompunere; obiectul este implementat prin construirea la un nivel superior a unor obiecte sau a unor clase de obiecte existente de nivel scazut. In timpul etapei de prototip al sistemului, dezvoltatorul poate alege sa intarzie implementarea tuturor obiectelor pana intr-un moment intarziat si acesta se poate baza pe specificatiile obiectului (cu implementari corespunzatoare) pentru a experimenta cu arhitectura si comportamentul sistemului. Similar, dezvoltatorul poate alege sa incerce mai multe reprezentari alternative asupra vietii obiectului, pentru a experimenta cu comportamentul implementarilor diferite.

Tranzitia între aceste etape de dezvoltare ar trebui, în mod ideal, să nu fie notificate, cu notatii compatibile utilizate în fiecare etapă. Mutarea la următoarea etapă implică rafinarea etapei anterioare prin adăugarea de detalii la clasele de obiecte existente si elaborarea de noi clase de a oferi functionalităti suplimentare. In timp ce informatia este integrata în obiecte, deciziile detaliate de proiectare despre reprezentarea datelor poate fi amânată până când sistemul este implementat. În unele cazuri, deciziile privind distribuirea obiectelor si dacă obiectele pot fi secventiale sau concurente pot fi, de asemenea, întârziate.

Acest lucru înseamnă că dezvoltatorii de software pot concepe modele care pot fi adaptate pentru diferite medii de executie. Acest lucru este exemplificat de abordarea arhitecturii bazate pe model (ABM), care propune că sistemele ar trebui să fie în mod explicit proiectate pe două niveluri, un nivel independent de implementare si un nivel dependent de implementare. Un model abstract al sistemului este conceput la nivelul independent de implementare, iar acest lucru este mapat la un model dependent de platforma mai detaliat care poate fi folosit ca bază pentru generarea de cod.

Compararea dezvoltarii obiect-orientate cu dezvoltarile traditionale

Sisteme orientate-obiect sunt mai usor de schimbat decât sistemele dezvoltate folosind alte abordări, deoarece obiectele sunt independente. Ele pot fi întelese si modificate ca entităti de sine stătătoare. Schimbarea implementarii unui obiect sau adăugarea de servicii nu ar trebui să afecteze alte obiecte de sistem. Deoarece obiectele sunt asociate cu lucrurile, există adesea o mapare clară între entităti din lumea reală (cum ar fi componente hardware) si obiectele lor de control din sistem. Acest lucru îmbunătăteste inteligibilitătii si, prin urmare întretinere a designului.

Obiectele sunt, in principiu, componente reutilizabile, deoarece acestea sunt incapsulari independente ale starii si a operatiilor. Modele pot fi dezvoltate folosind obiecte care au fost create în desene sau modele anterioare. Acest lucru reduce costurile de proiectare, programare si validare. Aceasta poate duce, de asemenea la utilizarea de obiecte standard (deci îmbunătătirea inteligibilitătii design-ului) si reduce riscurile implicate în dezvoltarea de software.

Dezvoltarea obiect-orientata a aplicatiilor este o metoda de ciclu de viata partiala. Aceasta se concentreaza pe stagiile de design si implementare a dezvoltarii software. Desi pasii urmati in formalizarea strategiei par mecanici, aceasta nu este o procedura automata, necesitand un efort major de cunostinte asupra lumii reale si intelegere intuitiva a problemei. Este necesara cuplarea dezvoltarii obiect-orientate a aplicatiei cu cerinte corespunzatoare si metode de analiza pentru a ajuta la crearea modelului de realitate al problemei.

Sistemele proiectate intr-o maniera obiect-orientata tind sa arate caracteristici diferite fata de cele proiectate intr-o maniera traditionala functionala. Sistemele obiect-orientate mari tind sa creeze straturi de abstractizare, unde fiecare strat denota o colectie de obiecte si clase de obiecte cu vizibilitate restrictionata fata de celelate straturi. Astfel de colectii se numesc subsisteme. Componentele care formeaza un subsistem tind sa fie plate din punct de vedere structural, decat sa fie strict ierarhice si profund imbricate.

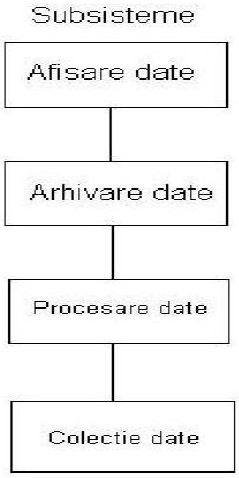

Figura 2 : Arhitectura pe straturi (subsisteme) a unei aplicatii OO

Stratul de afisare a datelor este stratul in care obiectele se ocupa cu pregatirea si prezentarea datelor intr-o forma inteligibila pentru om.

Stratul de arhivare a datelor este stratul in care obiectele se ocupa cu stocarea datelor pentru procesari ulterioare.

Stratul de procesare a datelor este stratul in care obiectele se ocupa cu verificarea si integrarea datelor colectionate.

Stratul de colectare a datelor este stratul in care obiectele se ocupa cu achizitionarea datelor din diverse surse.

Controlul global al fluxului intr-un sistem obiect-orientat este suficient de diferit fata de cel al unui sistem care se poate descompune functional. In cazuri ulterioare, exista tendinta de a exista un singur fir de control care urmareste liniile de descompunere ierarhice. In cazul unui sistem obiect-orientat, deoarece obiectele pot fi independente si autonome, nu se poate identifica un fir central de control. Pot exista mai multe fire active simultan in cadrul sistemului. Acest model nu este negativ, deoarece poate reflecta abstractizarea realitatii mai bine. Profilul apelurilor de subprograme intr-un sistem obiect-orientat consta in apeluri profund imbricate. Implementarea unei operatii a unui obiect consta des in invocarea operatiilor altor obiecte.

Exista multe beneficii care pot fi derivate din proiectarea obiect-orientata. Multi programatori observa doua realizari majore: primul este reducerea costului ciclului de viata software total prin cresterea productivitatii programatorului si reducerea costurilor de mentenanta, iar al doilea este implementarea sistemelor software care pot rezista la incercari malicioase si accidentale de corupere. Complexitatea, completitudinea si timpul de proiectare sunt in avantajul proiectarii obiect-orientate a unui produs software. Din punct de vedere al mentenabilitatii sistemelor obiect-orientate, datorita modularitatii software-ului, modificarile si extinderile sunt mult mai usor de operat decat in cazul programelor structurate intr-un mod traditional, orientat catre proceduri. Intelegerea si mentenabilitatea sunt crescute datorita faptului ca obiectele si operatiile lor sunt usor de localizat.

Cel mai important beneficiu al dezvoltarii aplicatiilor obiect-orientate este cel al folosirii unui mecanism de formalizare al modelului realitatii. Astfel se poate face o corespondenta directa si naturala intre lume si modelul ei abstractizat, care poate fi aplicata si in cazul problemelor care contin concurenta naturala. Metodele de dezvoltare obiect-orientata a aplicatiilor produc solutii mai bune pentru probleme care necesita procesare in timp real.

Exemplu de implementare clasei Salariat din figura 1:

```
class Salariat extends Persoana {

private String functie;
private int salariu;
private int ID;

public void setFunctie(String f, int id)
{       for(int i=1;i<=length(Salariat.ID))
{
if(Salariat.ID==id)
Salariat.functie=f;
}
}

public String getFunctie(int id)
{       for(int i=1;i<=length(Salariat.ID))
{
if(Salariat.ID==id)
return Salariat.functie;
}
}

public void setSalariu(int sal, int id)
{       for(int i=1;i<=length(Salariat.ID))
{
if(Salariat.ID==id)
salariu=sal;
}
}

public int getSalariu(int id)
{       for(int i=1;i<=length(Salariat.ID))
{
if(Salariat.ID==id)
return Salariat.salariu;
}
}

public void setID(int id)
{       Salariat.ID=id;      }

public int getID()
{       return Salariat.ID;  }       } //Salariat
```

Concluzii

Etapa esentiala in desfasurarea acestui proces este etapa de proiectare a sistemului, chiar daca este evident ca structura interna a acestuia este nesemnificativa pentru utilizator. S-a observat ca succesul aplicatiilor obiect-orientate depinde in principal de doua conditii:

1. Prezenta unei perspective arhitecturala legata si concreta

Arhitectura unui sistem obiect-orientat trebuie sa fie simpla si sa includa atat structura claselor si comunicarea dintre obiecte, cat si divizarea aplicatiei in module si nivele de abstractizare. Cateva conditii pentru crearea unei arhitecturi valide:
- nivele de abstractizare bine definite;
- interfetele claselor bine definite, cu schimbari minime ale interaciunilor in caz de modificare;
- schimbarea implementarii unei clase nu provoaca schimbari in interfetele sau implementarile celorlaltor clase;

2. Parcurgerea unui ciclu de dezvoltare combinat iterativ si incremental coordonat

Exista doua tipuri de cicluri de dezvoltare:
- ciclul de dezvoltare nedeterminat, in care este imposibil de stiut durata de dezvoltare a unui sistem, viteza dezvoltarii, termenul de finalizare, etc.Acesta implica costuri mari deoarece nu se cunoaste eficenta implementarii in mod concret.
-ciclul de dezvoltare cu reguli concrete pentru toate aspectele dezvoltarii, care inhiba capacitatea de creare si experimentare a echipei, dar care produce un produs sofware de calitate. Comunicare dintre programatori si utilizatori este ingreunata din cauza cerintelor si de aceea, costurile sunt crescute.

In realitate, nu exista o utilizare singulara a unui ciclu. Pe parcursul dezvoltarii aplicatiilor software, cele doua tipuri de cicluri de dezvoltare sunt combinate, existand un echilibru inclinat spre unul dintre ele in functie de conducatorii proiectelor si deciziile lor. Concluzia este ca ciclul de dezvoltare obiect-orientat al aplicatiilor ideal este o combinatie dintre ciclul iterativ si cel incremental.

Bibliografie

1. Ian Sommerville - *Software Engineering*
2. Grady Broch – *Object Oriented Development*
3. Zhiming Liu - *Object-Oriented Software Development with UML*
4. Proiectarea si dezvoltarea de aplicatii orientate obiect : http://www.biosfarm.ro/~dragos/papers/C++-Book/