Dezvoltare Software

folosind

modelul Obiect Orientat

Student: Popescu Andrei Alexandru

Grupa: 441 A

Materia: Ingineria Sistemelor

Profesor Coordonator: Stefan Stancescu

Cuprins

1. Introducere
- 1.1 Obiectul
- 1.2 Clasa
- 1.3 Instanta

2. Concepte cheie
- 2.1 Incapsularea
- 2.2 Mostenirea
- 2.3 Polimorfismul

3. Aplicarea conceptului obiect-orientat
- 3.1 In programare
- 3.2 In analiza si design
- 3.3 In specificatii
- 3.4 In interfetele cu utilizatorul

4. Avantajele si problemele ridicate de abrdarea obiect-orientata
- 4.1 Avantaje
  - 4.1.1 Reutilizare
- 4.2 Probleme

5. Concluzii

Bibliografie

Dezvoltare Software folosind modelul Obiect Orientat

1. Introducere

Modelul obiect-orientat este o abordare noua si diferita in domeniul dezvoltarii software. Cu toate ca conceptele sale sunt foarte naturale si apropiate de comportamentul si modul de gandire uman , aceste concepte au inceput sa fie aplicate relative tarziu in acest domeniu. Principiile obiect-orientat sunt vechi de mii de ani , oamenii folosing acest gen de gandire , avand trei metode de organizare tipice acestui mod de gandire:

- Se face diferenta intre un obiect si atributele sale ; de exemplu se face diferenta intre un copac si marimea acestia .
- Se face diferenta intre un obiect intreg si componentele sale ; de exemplu componentele unui copac sunt crengi si frunze.
- Crearea si distinctia intre clase de obiecte ; de exemplu se formeaza o clasa a tuturor copacilor si o clasa a tutuor pietrelor , facandu-se diferenta intre ele , deci facandu-se diferenta intre clase.

Dintre toate modelele de abordare in dezvoltarea software , anume : modelul procedural si modelul axat pe fluxul de date si modelul obiect-orientat , cel din urma , obiect-orientat este cel mai natural dintre toate si nu incearca inserarea de concept artificiale in procesul de dezvoltare software (analiza , proiectare , implementare).

1.1 Obiectul

Discutiile pentru definirea diverselor elemente din abordarea obiect-orientata este inca una in plina desfasurare , existand o multitudine de definitii fix din nevoia ca acestea sa fie cat mai natural si mai apropiate ce comportamentul uman. O definitie aproximativa a obiectului ar putea fi:

Un obiect este abstractizarea a ceva ce exista in lumea reala sau in mintea noastra , care face parte din sistemul pe care vrem sa il modelam si despre care vrem sa avem informatii. Un obiect ar putea si un client, un sensor de temperature sau o stiva. De altfel acesta nu este decat un simplu sir de date. Insa in acest concept procedurile si datele trebuie sa faca parte dintr-un intreg . Astfel definitia de mai sus poate fi reformulate in :

Un obiect este abstractizarea a ceva din lumea reala , palpabil, sau ceva imaginar , care este definit prin atributele sale dar si de actiunile care pot fi facut asupra lui sau de catre el.

Acesta este conceptul de baza al gandirii obiect-orientate. Datele nu vor mai fi separate de procedure , acest concept incercand sa le apropie cat mai mult. Obiectele din lumea reala vor fi reprezentate abstractizat in modele ca o unitate ce contine attribute si activitati. De exemplu daca am dori sa reprezentam o pisica , vom stoca caracteristicile ei cum ar fi culoarea ochilor , lungimea cozii dar si actiunile ei , precum torsul si bautul de lapte. Totusi aceste concept nu ar trebui sa fie aruncate undeva aiurea in descrierea programului , ci vor fi associate cu obiectul pisica , altfel atributele sau actiunile neavand nici-o noima fara asocierea cu un obiect.

1.2 Clasele

Clasificarea este o actiune pe care oamenii au facuto si o vor face intotdeauna pentru a putea stapani complexitatea si numarul imens de obiecte din mediul inconjurator. De exemplu daca avem doua entitati total diferite , ar trebui sa atribuim pentru fiecare entitate attribute respective actiuni specifice. Clasele permit o clasificare a entitatilor , daca doua entitati sunt in aceeasi clasa vom sti ca impart aceleasi attribute si actiuni. Acest lucru nu inseamna inca ca

toate atributele si actiunile trebuie sa fie identice , acestea fiind doar partial asemanatoare.

O definitie pentru clase ar putea suna asa:

O clasa reprezinta un set de obiecte care au in comun anumite proprietati/atribute.

Aceste lucruri ne amintesc de exemplu de programare si limbajele de programare , unde doua variabile de tip integer au acelasi tip dar contin doua numere diferite. De altfel o clasa este un tip in marea majoritate a limbajelor de programare.

In momentul in care o clasa a fost abstractizata in lumea reala se vor identifica atributele si metodele ce apartin acestei clase. O metoda este denumita o procedura ce apartine unui obiect , o alta denumire pentru aceasta de exemplu in limbajele de programare C++ este functie membru. Atributul este informatia pe care dorim sa o stocam despre obiectul /obiectele unei clase , acestea mai sunt numite si date membru.

In matematica conceptul de clasa are diverse relatii de echivalenta , de exemplu o clasa poate fi impartita in doua , iar reuniunea acestora din urma va rezulta prima clasa impreuna cu toate valorile sale. Din pacate in gandirea obiect-orientata acest concept de relatii de echivalenta nu se poate aplica de cele mai multe ori. Acest lucru se datoreaza faptului ca criteriile de includere in clase pot fi intelese diferit , de exemplu un electro-motor poate fi atribuit clasei de obiecte care functioneaza cu electricitate dar si clasei de obiecte ce pune in miscare alte obiecte. Nu este intotdeauna clara o identificare a claselor si o atribuire corecta a obiectelor la clase.

1.3 Instantele

Instanta unei clase este un element al setului ce reprezinta clasa. Foarte multa lume confunda/egalizeaza conceptele de obiect si instanta.

De cele mai multe ori acest lucru este suficient , dar exista si cazuri in care nu putem confunda conceptele de tip si clasa cum nu se poate confunda/ trebuie sa se faca o diferenta intre conceptul de obiect si conceptul de instanta. Putem spune ca obiectul este prezinta un lucru in lumea reala pe cand o instanta este doar relevanta in abstractizarea aplicatiei curente.

Cazul evident in care un obiect si instanta corespunzatoare lui nu pot fi considerate echivalente este cand instanta este data drept un parametru / o valoare pentru o procedura. Astfel putem spune ca o instanta este o implementare ce poate fi aplicata doar in stadiul de programarea codului (etapa de implementare).

2. Concepte cheie

Diferiti autori ofera o multitudine de criterii care ar trebui sa fie atribute/concepte cheie pentru abordarea obiect-orientata. Pentru grupul de management al obiectelor de exemplu , identitatea obiectelor are o importanta deosebita, adica un obiect poate fi identificat printr-o cheie , independent de atributele sale. Exista , insa o serie de concepte care apar aproape este tot , la toti autorii , avand rolul de a face diferenta intre sistemele orientate-obiect si non-orientate-obiect .

Alte concepte regasite la aceast nivel nu afecteaza numai abordarea obiect-orientata. Un exemplu de astfel de concept este relatia dintre doua obiecte de tipul intreg-componenta sau o relatie care poate fi intotdeauna exprimata intre

doua obiecte deoarece pana la urma obiectul abstractizat este doar un obiect de date (impreuna cu operatiile/aplicatiile sale).

2.1 Incapsularea

La inceput , programatorii trebuiau sa utilizeze numai modeul procedural; foloseau cel mai bun algoritm pentru rezolvarea problemei. Programele creeate nu aveau nici-un fel de structura de organizare. Mai tarziu procedurile au fost introduse in module functionale , dar de cele mai multe ori o procedura dint-un modul nu avea nicio legatura cu o alta procedura din acelasi model. Urmatorul pas a fost abstractizarea datelor , proces numit incapsulare .

Datele au fost grupate cu procedurile in care erau utilizate. In mod ideal exista o singura interfata prin care datele sunt acesate de mai multe proceduri , fara a exista o interfata care ofera acces direct la date, acest concept numindu-se ascunderea datelor. Procedura nu ar trebui sa arate modul in care a manipulat datele , intregul concept fiind numit ascunderea informatiilor.

Cand datele sunt mai mult de o adunatura de iteme fara nicio legatura , si chiar prezinta un concept coerent , rezulta un tip abstract de date numit ADT. Majoritatea limbajelor de programare incearca sa implementeze conceptul de ascunderea a informatiilor insa nu toate reusesc si o implementare eficienta a ideilor de obiect-orientat.

2.2 Mostenire

O planta este un oraginsm. Un copac este o planta. Un stejar este un copac. Ierarhii exista peste tot in lume. Este posibil sa construiesti ierarhii de obiecte. O clasa inferioara alteia in ierarhie, avand toate atributele si metodele clasei ce se afla mai sus in ierarhie, inseamna ca aceasta va mosteni clasa superioara.

Clasa ce se afla mai sus in ierarhie se numeste clasa de baza sau superclasa (clasa parinte), iar cea care mosteneste se numeste subclasa sau clasa derivata. Cele doua clase se afla intr-o relatie, putem spune ca o clasa derivata este un fel de superclasa. Acest lucru inseamna ca superclasa este mai generala (un arbore), in timp ce clasa derivata este o specificatie a clasei de baza (un stejar-arbore). De aceea acest tip de relatie este uneori denumita relatie de tip generalizare-specializare. Mostenirea poate fi folosita pentru a mapa ierarhii din lumea reala apropiate de model. Adesea, clasele de baza nu sunt una dintre entitatile pe care le cautam in aplicatie. De exemplu, consideram o stiva si o coada. Acestea sunt obiecte ale imaginatiei noastre si vrem sa construim clase pentru ele. Dupa cum se dovedeste, acestea au multe in comun: se pot pune elemente in ele si se pot prelua elemente din ele. Deci, este posibil sa definim o clasa container, care va reprezenta atributele comune dintre cele doua clase, si vom face stiva si coada subclase ale containerului. Deoarece, in mod normal clasa container este prea abstracta pentru a fi utilizata, aceasta se va numi superclasa abstracta si nu se vor putea crea instante de obiect.

Anumite limbaje permit mostenirii multiple, de exemplu o clasa poate mosteni mai multe superclase, pe cand alte limbaje nu suporta mostenitii multiple. Nu este clar daca mostenirea multipla este sau nu necesara sau pur si simplu convenabila.

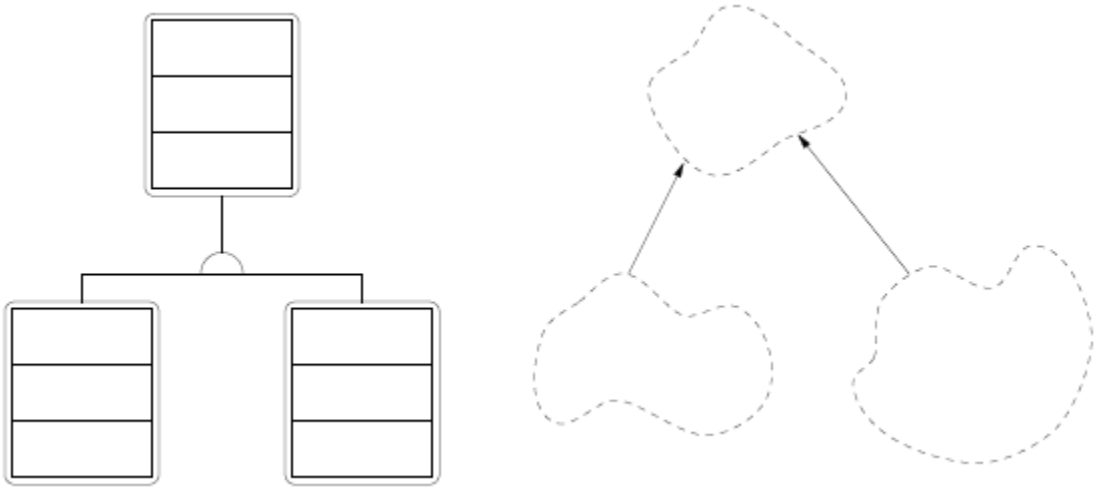

Figura 1. Notatii mostenire pentru metodele Coad-Yourdon si Booch

Figura 1 prezinta doua notatii din multitudinea de notatii diferite pentru mostenire. De retinut ca sageata indica direct clasa de baza, care este oarecum contraintuitiv la prima vedere, dar are sens deoarece, de obicei, o clasa derivata stie care este clasa de baza (care sunt clasele de baza), dar o clasa de baza nu are cunostinte despre clasele sale derivate.

2.3 Polimorfismul

Polimorf este derivat din greaca si inseamna multe forme. Cand ceva este polimorf poate lua diferite forme.

Unii autori nu vad acest lucru ca un element cheie, dar este foarte puternic si fara el orientarea pe obiecte nu poate fi realizata integral.

Polimorfismul este un concept acceptat atunci cand putem folosi o clasa unde o alta este preconizata. Acest lucru este mai greu de realizat atunci cand clasa folosita in acest context are cel putin o caracteristica comuna cu clasa care trebuia folosita. Totusi, in scopul de a evita confuzia (si de a imbunatati performanta unui limbaj de programare orientata pe obiect) polimorfismul este adesea folosit impreuna cu mostenirea: polimorfisumul este atins atunci cand este posibil sa se utilizeze o clasa derivata in locul acelei clase.

Luam in considerare un exemplu standard: o ierarhie de obiecte geometrice, ca patrate, triunghiuri, cercuri, etc. Acestea sunt toate clase derivate ale unei clase abstracte numita forma. Toate au multe atribute in comun, in special, toate se pot desena. Avand in vedere o colectie de forme, sarcina de a le desena poate fi realizata dupa cum urmeaza: (sintaxa C++)

```
for (i=0; i<ShapeCollection.length; i++)
{
        if (ShapeCollection[i].type == square)
drawSquare(ShapeCollection[i]);
        else if (ShapeCollection[i].type == circle)
drawCircle(ShapeCollection[i]);
        else if . . .
```

```
}
```

Aceasta sintaxa este ciudata si predispusa la eroare. Ea trebuie sa fie repetata peste tot in program oriunde obiectul este utilizat. La introducerea unei noi clase, tot codul sursa trebuie sa fie schimbat. Este mai corect sa se scrie:

```
for (i=0; i<ShapeCollection.length; i++)
        ShapeCollection[i].draw;
```

La urma urmei, obiectele stiu cum sa se deseneze singure. Vom folosi toate obiectele care sunt derivate din clasa forma, ca si cand ar fi de fapt un sablon. Bineinteles, clasa forma va trebui sa aiba o metoda de desenat (draw) pentru a putea compila. Cu toate acestea, ce functie va fi apelata se va decide la momentul executiei, acest concept fiind cunoscut drept legare dinamica.

Conceptul de polimorfism este ceea ce face mostenirea puternica, pentru ca acesta face uz de caracterul comun exprimat de relatia de mostenire.

3. Aplicatii ale conceptului

Unele concepte orientate pe obiect sunt cunoscute de zeci de ani fara a fi mentionate ca orientate pe obiect. Este de remarcat ca metodele orientate pe obiect nu au aparut peste noapte, dar au evoluat incet pana au ajuns la popularitatea lor actuala.

Multe produse din zilele noastre sunt etichetate ca obiect orientate, dar, bineinteles, nu sunt toate. In urmatorul paragraf vom observa cat de usor este de aplicat conceptul de obiect orientat.

3.1 Programarea

Totul a inceput la sfarsitul aniilor 1960 cand primul limbaj de programare orientat pe obiect (LPOO) a fost dezvolat: Simula 67. A fost nevoie, totusi, de 20 de ani pana cand ideiele au fost in utilizare pe scara larga. Smalltalk, unul din cele dou cele mai importante LPOO-uri, a devenit obiect-orientat in versiunea 76.

Un LPOO suporta toate conceptele cheie enumerate in sectiunile anterioare. Daca un anumit limbaj de programare nu le suporta, atunci acesta nu este orientat pe obiect.

Aceste concepte permit o perfectionare a tuturor tehnicilor de programre structurata cunoscute pana atunci. In plus, acestea ofera posibilitati, cum ar fi poliformismul, care au fost anterior imposibile sau foarte stangaci realizate. Multe limbaje permit supraincarcarea procedurilor si a operatorilor, permitand astfel definirea unei clase cu un set complet de operatii, astfel incat acestea sa fie utilizate exact ca tipurile incluse.

Programarea poate profita din plin de suportul obiect-orientat. Cu toate acestea,daca in faza de analiza si proiectare (design) se produce o abstractie procedurala, atunci utilizarea unui LPOO este limitata; pentru a nu pierde oportunitati importante este de preferat sa se realizeze un design orientat pe obiect.

3.2 Analiza si design

Analiza si design-ul obiect-orientate au fost dezvoltate chiar mai tarziu decat programarea orientata pe obiect. Acum exista diferite metode pentru ambele etape.

Este evident ca atunci cand design-ul nu produce o structura obiect-orientata, puterea unui LPOO va fi irosita. Este important sa se identifice clasele in etapa de proiectare si sa nu fie luate ca o problema de implementare. Poate fi realizat un design foarte apropiat de sistemul dorit, prin urmare, mai usor de inteles. Un design in care erorile sunt mult mai usor de reperat, si atunci cand acestea vor fi eliminate sa nu afecteze decat o mica parte a design-ului, intrucat obiectele exista dar sunt slab cuplate.

Cand cineva are ca scop realizarea unui design obiect-orientat, este avantajos sa vizualizeze sistemul si sarcina ca obiect-orientat de la inceput si sa analizeze sistemul folosind tehnici obiect-orientate. Ar trebui, totusi, sa fie remarcat ca ordinea in care se aplica conceptul de obiect-orientat in acest articol nu corespunde exact cu ordinea in care activitatile vor fi incepute in timpul

dezvoltarii software, dar reflecta in principal nivelul de intelegere a acestei activitati in zilele noastre. In timp ce programarea orientata pe obiecte este acceptata si utilizata pe scara larga, analiza este de cele mai multe ori efectuata in moduri mai conventionale. Este de sperat ca acest lucru sa se schimbe in viitorul apropiat.

3.3 Specificatii

Metode formale sunt acceptate ca mijloace pentru a produce specificatii clare, de fapt singurele mijloace de a le produce. Ele castiga popularitate, deoarece acestea produc un software mai de incredere, care este tot mai dorit in lumea noastra computerizata.

Multe din aceste metode insa au o problema foarte mare , ele sunt foarte greu de inteles mai ales cand proiectele sunt din ce in ce mai mari. Exista diferite abordari pentru a rezolva aceasta problema. Una dintre aceste metode este integrarea , este incercarea de a combina metode formale cu metode structurate ; acestea au avantajul ca sunt foarte usor de inteles pentru clienti. Cealalta metoda este de a extinde metoda formala curenta cu elemente obiect-orientate. Aici conceptele obiect-orientate sunt folosite pentru a simplifica problema , pentru a o imparti in mai multe parti , si rezolvarea acestora pe rand.

In mod evident cele doua abordari nu sunt in antiteza , ci se complementeaza astfel sunt posibile si combinatii de metode neformale cu metode formale bazate pe concepte obiect-orientate.

3.4 Interfata cu utilizatorul

Interfata cu utilizatorul a fost de cele mai multe ori legata de conceptele obiect-orientate. Uneori acest lucru a fost folosit fals de catre dezvoltator pentru a-si putea vinde produsul , interfata cu utilizatorul neutilizant ideile obiect-orientate . In ciuda acestui fapt naturaletea limbajului obiect-orientat si-a pus amprenta si pe creearea interfetelor cu utilizatorul.

Putem lua un exemplu din foile de calcul (excell). La selectia mai multor celule la o interfata conventionala operatiunea dorita ar fi trebuit selectata din meniul principal , la abordarea obiect-orientata se poate creea un meniu contextual (afisarea lui se poate face de exemplu prin right-click) , meniu care are exact operatiunile aplicabile pentru celulele respective. Un alt exemplu de acest gen ar putea fi asocierea unei celule cu un editor de formule matematice folosind abordarea obiect orientata. Exact acesta este scopul modelului obiect-orientat : sunt prezente in acelasi loc nu doar datele vizibile ale unui tip de entitate , ci si procedurile folosite .

4. Avantajele si problemele ridicate de abordarea obiect-orientata

4.1 Avantaje

Foarte multa lume a vazut abordarea obiect-orientata ca o rezolvarea a tuturor problemelor ridicate de software development , dar nu a fost asa , deoarece acest domeniu este foarte complex so este foarte putin probabil ca acest gen de schimbare sa aduca beneficii foarte mari la nivel de productivitate. Insa abordarea obiect-orientata a fost un real progres fata de vechile tehnici de dezvoltare .

Conceptele de obiect-orientat , sunt mai naturale , mai usor de formulat si sunt mult mai apropiate de problema sau sistemul pe care trebuie sa il modeleze. Structura problemei poate fi reprezentata cat mai direct , deci poate fi simplificata , astfel poate fi inteleasa mai usor iar erorile vor fi mai putine.

Deoarece notiunea de obiect-orientat este aceeasi in toate stadiile de creeare a unui produs software, se poate folosi aceeasi terminologie si aceleasi notatii pe tot parcursul creatiei. Acest lucru creste de asemenea gradul de intelegere si simplitatea problemei .

Cel mai importat avantaj al abordarii obiect-orientate este reutilizarea. Datorita incapsularii foarte puternice asociate cu conceptul obiect-orientat , reutilizarea secventelor de program devine triviala.

4.1.1 Reutilizarea

Reutilizarea semnifica utilizarea unei componente care a mai fost folosita. Cea mai simpla forma a utilizarii este utilizarea de secvente din cod (copy&paste) din alte programe. Orientarea pe obiecte faciliteaza reutilizarea a intregi clase, iar daca acestea nu sunt exact pe placul programatorului acesta poate oricand sa le modifice comportamentul prin mostenire. In mod ideal ar trebui sa fie posibil sa se poate reutiliza toate rezultatele obtinute in urma etapei de analiza si a etapei de design. La reutilizare se pot folosi aceste date si se fac doar mici adjustari pentru a reflecta noile cerinte.

Se presupune ca reutilizarea a crescut eficienta si productivtatea foarte mult , insa in realitate aceasta crestere nu este chiar asa de dramatica . Cresterea exista si este considerabila insa statisticile arata ca aceasta crestere a productivitatii nu este atat de mare pe cat se presupune.

4.2 Probleme

Cea mai mare problema a conceptului obiect-orientat pare a fi tranzitita de la metodele mai vechi. Pot exista domenii in care celelalte metode de abordare poti fi utilizate si sunt utilizate mai mult din inertie.

5. Concluzii

In final se poate spune ca conceptul obiect-orientat a revolutionat industria dezvoltarii de software . Nu numai ca productivitatea a crescut considerabil , a mai crescut si gradul de accesibilitate , usurinta in care clientul isi poate formula problema pentru ca aceasta sa poata fi implementata cat mai rapid si eficient.

Astfel metoda de dezvoltare bazata pe obiecte-orientate este mai buna din toate puntele de vedere fata de predecesoarele ei.

Bibliografie

1. Edward V. Berard. *Essays on object-oriented software engineering*
2. R. G. G. Catell. *Object Data Management: Object-Oriented and Extended Relational Database Systems*.
3. Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach*.
4. Stuart M. Creeley. *The object oriented approach. Its concepts, a development methodology and software reuse*.
5. Peter Coad and Edward Yourdon. *Object-Oriented Analysis*.
6. Peter Coad and Edward Yourdon. *Object-Oriented Design*.