CURS 1

PAGINA 2

A gandi “orientat obiect”

PAGINA 3

Evolutia programarii Cod masina Limbaj asamblare Limbaje independente de masina Proceduri si functii Obiecte Servicii

PAGINA 4

Istoric ~1960 SIMULA 1 (1962) si Simula 67 (1967), Norwegian Computing Center in Oslo, Norway Smaltalk (1970) – XEROX "C with Classes", predecesorul C++, 1979, Bjarne Stroustrup 1995, Java dezvoltat de James Gosling la Sun Microsystems. 1980 – Revista Byte Prima conferinta dedicata OOP, Portland, Oregon, USA

PAGINA 5

Generalitati  POO a evoluat din programarea structurata (PS)  POO este: O idee revolutionara O evolutie normala  Aplicabilitate POO: sisteme distribuite PS: fluxuri de date, descompunere de procese

PAGINA 6

Limbaj si gandire (1) M. Sapir (1956): “Oamenii se gasesc la cheremul unui limbaj particular care reprezinta mediul de comunicare in cadrul unei societati”  Limbajul influenteaza gandirea d.e. vocabularul extensiv (eschimosii si zapada) Concluzie: un limbaj OO poate simplifica dezvoltarea unei solutii OO

PAGINA 7

Limbaj si gandire (2) Exemplu: analiza secventei DNA: fie un vector cu N variabile in cadrul careia trebuie determinata o secventa repetitiva de lungime M Rezolvare intr’un limbaj standard: M x N x N operatii Rezolvare intr’un limbaj OO M x N x log(N) operatii Observatie: Limbajul OO NU este mai rapid decat limbajul structurat

PAGINA 8

Ipoteza lui Church & postulatul lui Whorf Ipoteza lui Church / Church-Turing (1936): orice rationament pentru care exista o “procedura efectiva” (i.e. are la baza un algoritm) poate fi realizat de o masina Turing. Reguli: • Numarul de instructiuni trebuie să fie finit. • Iesirea ar trebui să fie obtinuta dupa parcurgerea unui numar finit de pasi. • Instructiunile pot fi implementate în lumea reala. • Instructiunile nu ar trebui sa necesite o intelegere complexa. <https://www.youtube.com/watch?v=0D7yInuKvKs>

PAGINA 9

Ipoteza lui Church & postulatul lui Whorf Postulatul lui Whorf / Sapir-Whorf (1956): este posibil ca un individ, ce comunica intr’un limbaj, sa nu poata comunica ganduri / idei intr’un alt limbaj Astfel, se postuleaza faptul ca natura unei anumite limbi influenteaza gandirea obisnuita a vorbitorilor ei. Diferitele modele de limbaj genereaza diferite modele de gandire. Aceasta idee contesta posibilitatea reprezentarii perfecte a lumii cu limbajul, deoarece recunoaste ca mecanismele oricarei limbi conditioneaza gandurile comunitatii de vorbitori ai acesteia. <https://study.com/academy/lesson/sapir-whorf-hypothesis-examples-anddefinition.html>

PAGINA 10

Consecinte Pentru Ipoteza lui Church Orice limbaj in cadrul caruia poate fi simulata o masina Turing, este suficient de performant pentru a permite implementarea oricarui algoritm Toate limbajele de programare sunt identice Pentru Postulatul lui Whorf: Pot exista idei ce pot sa fie exprimate mai bine intr’un limbaj decat intr’altul Consecinta George Steiner (1975 - After Babel): orice act de comunicare umana poate fi vazut ca implica o traducere

PAGINA 11

O noua paradigma Programul OO creaza obiecte ce incapsuleaza date si procedure ce vor actiona asupra acelor date Un POO este structurat ca o comunitate de agenti / obiecte ce interactioneaza intre ele Fiecare obiect produce un serviciu sau realizeaza actiuni Exemplul florarului

PAGINA 12

Caracteristicile POO O noua metoda de proiectare a aplicatiilor pornind de la o reprezentare naturala Programele sunt gandite a avea componente ce functioneaza independent, fiecare cu propriile proprietati si functionalitati Un sistem este proiectat ca o interconexiune de asemenea componente.

PAGINA 13

Caracteristicile programarii structurate Programele sunt gandite strict secvential Se utilizeaza branching conditions in cazul implementarii deciziilor Se utilizeaza functii / proceduri pentru implementarea secventelor repetabile. Sinteza: programele definesc date, dupa care apeleaza subprograme pentru a prelucra acele date

PAGINA 14

Limitarile programarii structurate In viata reala evenimentele nu sunt intotdeauna secventiale Nu este simplu de gasit solutii secventiale pentru problemele reale Codul aplicatiei nu poate fi re-utilizat si depanat independent Codul nu se poate “adapta” la situatiile in schimbare  Abordari OO sunt benefice in cadrul aplicatiilor de dimensiuni mari (>1000 linii de cod)

PAGINA 15

Actiunile sunt initiate in POO prin transmitere de mesaje de catre un agent (obiect) responsabil pentru o actiune Programele sunt gandite ca interactiuni / asocieri de obiecte In cadrul POO nu trebuie “anticipat” orice – ci trebuie doar utilizate serviciile obiectelor POO – un nou model de gandire

PAGINA 16

Programarea structurata

PAGINA 17

POO vs. Programarea structurata

PAGINA 18

Evolutia limbajelor de programare (1) Generatia I (1954 - 1958) – formule matematice Fortran I, Algol 58, IPL Generatia II (1959 -1961) – algoritmi Fortran II, Algol, Cobol Generatia III (1962 - 1970) – abstractizare date PL/I, Pascal, Algol 68, C, Fortran77 Limbaje OO Smalltalk, C++, Eiffel Limbaje actuale: Java, Python, C#

PAGINA 19

Evolutia limbajelor de programare (2)

PAGINA 20

Evolutia limbajelor de programare (3)

PAGINA 21

Evolutia limbajelor de programare (4) Generatia I Generatia II - Mecanisme de “transfer parametrii” - Dezvoltare stracturi de control – Subprograme

PAGINA 22

Evolutia limbajelor de programare (5) Generatia III Generatia IV

PAGINA 23

Evolutia limbajelor de programare (6)

PAGINA 24

Programarea functionala vs. Programarea procedurala vs. Programarea OO (1)

PAGINA 25

Programarea functionala vs. Programarea procedurala vs. Programarea OO (2)

PAGINA 26

Avantajele POO Analiza si proiectare mai facila Reutilizare cod O mai buna mentenanta a programelor Reducerea codului recurrent Mascarea informatiilor Robustete sporita O mai buna securitate a aplicatiilor

PAGINA 27

Concluzii … istorice “I think there is a world market for maybe five computers.” Thomas Watson, chairman of IBM, 1943 “Computers in the future may weigh no more than 1.5 tons.” Popular Mechanics, 1949 “There is no reason anyone would want a computer in their home.” Ken Olson, president, chairman and founder of Digital Equipment Corp., 1977

CURS 2

PAGINA 2

Conceptele de baza ale POO

PAGINA 3

Introducere Descompunerea sistemelor de mari dimensiuni: 1. algoritmic: pune in evidenta ordinea evenimentelor 2. Prin intermediul obiectelor: pune in evidenta elementele ce actioneaza / sunt supuse unor actiuni Diferente fata de abordarea clasica:  POO se bazeaza pe obiecte si nu pe algoritmi  Fiecare obiect este o instanta a unei clase  Clasele pot fi legate intre ele prin relatii de mostenir e Obs: daca una din aceste caracteristici lipseste, nu este vorba de programare OO

PAGINA 4

Stiluri de programare 1. Orientatate catre proceduri  algoritmi 2. Orientatate catre obiecte  clase si obiecte 3. Orientatate catre logica  scop (calcul predicativ) 4. Orientatate catre reguli  reguli de tip “if – then” 5. Orientatate catre constrangeri  relatii de invarianta

PAGINA 5

Principii de baza ale abordarii obiectuale

PAGINA 6

Ce este abstractizarea (1) ? Exemplul atlasului:  scara de reprezentare a marimilor Definitie: procedura de mascare a detaliilor unui obiect in scopul accentuarii / punerii in evidenta a altora specte / detalii / structuri Nivele de abstractizare: 1. Cel mai de sus: comunitatea obiectelor ce trebuie sa interactioneze unele cu altele in vederea indeplinirii unui scop comun

PAGINA 7

Ce este abstractizarea (2) ? 2. Valabil in anumite limbaje OO: grup de obiecte ce lucreaza impreuna si care sunt grupate in unit-uri, d.e. package (Java), name spaces (C++) si units (Delphi) 3. Nivelul de interactiune intre 2 obiecte (interactiune client / server, C/S). In acest caz exista 2 sub-nivele de abstractizare: Abstractizare client  server Abstractizare server  client

PAGINA 8

Ce este abstractizarea (3) ? Tipuri de abstractizari: entitate / actiune / ms virtuala / incident Abstractizarea este legata de notiunea de “invarianta”; ruperea unei invariante conduce la lipsa vizibilitatii unor obiecte cu altele Orice abstractizare are proprietati Statice, de ex nume fisier Dinamice, de ex valoarea unei prop statice  un nume de fisier poate fi schimbat Consecinta: abstractizarea si incapsularea sunt concepte complementare, i.e. pentru ca abstractizarea sa functioneze, trebuie sa existe incapsulare

PAGINA 9

Ce este incapsularea ? Ascunde implementarea fata de clienti  Clientii depind de interfata

PAGINA 10

Incapsulare vs abstractizare

PAGINA 11

Incapsulare vs abstractizare Utilizare • Abstractizare: rezolvă dificultatile ce apar în etapa de proiectare. • Încapsularea: rezolvă dificultatile ce apar în etapa de implementare. Focus • Abstractizare: focalizarea pe ceea ce face obiectul în loc de modul în care o face. • Încapsularea: permite ascunderea codului / datelor într’o singură unitate pentru a le securiza de lumea exterioară.

PAGINA 12

Incapsulare vs abstractizare Implementare Abstractizare: se utilizeaza folosind Interfața și Clasa abstractă. Încapsulare: se utilizeaza folosind Modificatori de acces (Public, Protejat și Privat.) Utilizare Abstractizare: ceea ce ar trebui făcut. Încapsulare: modul în care ar trebui făcut. Aplicabilitate Abstractizare: În timpul proiectarii. Încapsulare: În timpul implementarii.

PAGINA 13

Ce este polimorfismul ? descrie situațiile în care “ceva” se întâmplă în mai multe forme diferite. În informatică, polimorfismul descrie conceptul accesarii de obiecte de diferite tipuri prin aceeași interfață.

PAGINA 14

Ce este mostenirea ? Ierarhia reprezinta o ordonare a abstractizarii.

PAGINA 15

Concepte de baza ale POO Obiect Clasa Atribut Operatie Componenta / modularitate Mostenire Incapsularea Polimorfism

PAGINA 16

Obiecte (1) Un obiect reprezinta o entitate (fizica, conceptuala sau soft) 1.Entitate fizica 2.Entitate conceptuala 3.Entitate software Peste Piramide Componenta (subsistem)

PAGINA 17

Obiecte (2) Definitie formala: un obiect este un concept (real sau abstract) cu frontiere bine definite si posibilitate de actiune. Se caracterizeaza prin: Evolutie – influentata de istoricitatea obiectului (d.e. automatul pentru bauturi); reprez modul in care obiectele actioneaza / reactioneaza in termini de modificare de stare si schimb de mesaje Stare: orice obiect are o stare, i.e. ocupa un spatiu fizic de memorie Identitate unica: “propietatea unui obiect ce il face distinct fata de un alt obiect” (Copeland)

PAGINA 18

Starea unui obiect Starea unui obiect reprezinta una din conditiile posibile de existenta ale acestuia In mod normal, starea unui obiect se schimba in timp Este reprezentata de: Valorile atributelor + Instantele relatiilor (relationship instances ) Sofer Nume: Ion Popescu ID: 123456789 Data angajarii: 01.01.01

PAGINA 19

Evolutia determina modul in care obiectele (re)actioneaza la solicitarile altor obiecte. Evolutia unui obiect este reprezentata de totalitatea raspunsurilor la setul de mesaje primite (i.e. totalitatea operatiilor pe care un obiect le poate realiza) Observatie: orice obiect are o identitate unica (chiar daca starea sa este identica cu cea a unui alt obiect) Evolutia unui obiect (1) Sistem de inregistrareCurs 100 – Geometrie Conformitate Asignare curs

PAGINA 20

Operatii De modificare: altereaza starea unui obiect De selectie: operatia acceseaza starea unui obiect dar NU o altereaza De iteratie: operatia prin care se permite ca toate partile unui obiect sa poata fi accesate intr’o ordine predefinita Constructor: crearea unui obiect si/sau initializarea starii acestuia Destructor: elibereaza starea unui obiect si/sau il distruge Evolutia unui obiect (2)

PAGINA 21

Clase Ansamblul obiectelor cu aceeasi semantica, proprietati (atribute) similare si comportament comun Un obiect reprezinta instanta a unei clase Gruparea obiectelor in clase reprezinta rezultatul unei operatii de abstractizare: Evidentierea caracteristicilor comune Eliminarea elementelor nespecifice

PAGINA 22

Masina Atribute: Model Serie\_motor An\_fabricatie Operatii: Deplasare Reparare Oprire Clasa – exemplu

PAGINA 23

Clase & obiecte – exemplu © Faculty of Computer Science & Information System, UTMClasa “Arbore” Obiecte (instantieri ale clasei): arbori, cu atribut inaltime

PAGINA 24

Tipuri de clase Clasa abstracta. O clasa incompleta care defineste elementele comune a mai multor clase Fara instantiere. Clasa concreta O clasa completa Descrie complet un anumit concept Poate avea instantieri

PAGINA 25

Relatii intre obiecte si clase O clasa reprezinta o definitie abstracta a unui obiect Defineste structura si evolutia pentru fiece obiect ce apartine clasei respective Poate fi un template pentru crearea de obiecte Obiectele pot fi grupate in clase

PAGINA 26

Atribute & operatii Atributele reprezintă diferitele caracteristici ale clasei. Ele pot fi, de asemenea, denumite caracteristici ale clasei si pot diferenția o clasă de alte clase. Operatiile (metodele) sunt responsabile cu definirea / modificarea comportamentului unei clase și al obiectelor aferente acesteia.

PAGINA 27

Modularitate Modularitatea reprezinta procesul de descompunere a unei probleme (program) într’un set de module, astfel încât să reducă complexitatea generală a problemei. Modularitatea (Booch): „Modularitatea reprezinta proprietatea unui sistem care a fost descompus într-un set de module coezive și slab cuplate.” Modularitatea este intrinsec legată de încapsulare. Modularitatea poate fi vizualizată ca o modalitate de a mapa abstracțiile încapsulate în module reale, fizice, cu o coeziune ridicată în cadrul modulelor, iar interacțiunea sau cuplarea lor între module trebuie sa fie scăzută.

PAGINA 28

Caracteristici modularitate Exista programe (d.e. Smalltalk) ce nu contin module (i.e. clasa reprez. singura unitate fizica de descompunere) Scopul modularizarii: reducerea costurilor prin accesarea independenta a fiecarui modul  In general clasa + obiect se regasesc in module

PAGINA 29

Componenta Partea unui sistem ne-triviala si inlocuibila ce indeplineste o functie bine precizata in contextul unei arhitecturi bine definite. O componenta poate fi: Cod sursa O structura executabila

PAGINA 30

Proprietatea de partajare a atributelor si operatiilor unei / unor clase de catre alta / alte clase in cadrul relatiilor ierarhice, diferentele ramanand neschimbate Prin mostenire se pot exprima relatii de: Generalizare Specializare Clasificare Aproximatie Evolutie Tipuri de mosteniri: Simple Multiple Mostenire

PAGINA 31

Generalizare

PAGINA 32

Gruparea detaliilor de implementare intr’o zona ascunsa, inaccesibila din exterior Consecinta: reducerea gradului de interdependenta dintre elemente Incapsularea

PAGINA 33

Vizibilitatea claselor este structurata pe trei nivele: Public  toti! Privat  proprie unei clase Protejat  vizibilitate la nivelul subclasei Incapsularea

PAGINA 34

1. Dezvoltarea de componente software trebuie sa furnizeze utilizatorilor toata informatia necesara spre o buna utilizare a componentelor, dar NICI o alta informatie in plus. 2. Dezvoltatorul de componente software trebuie sa primeasca toata informatia necesara in vederea finalizarii cu succes a responsabilitatilor sale referitoare la componente, dar NICI o alta informatie in plus. Principiile lui David Parnas

PAGINA 35

Incapsulare – consecinte Obiectele incapsuleaza atat datele cat si procedurile aferente manipularii datelor

PAGINA 36

Capacitatea unui operator de a se aplica obiectelor din clase diferite Polimorfism (1) Deplasare Deplasare Deplasare

PAGINA 37

sau Abilitatea unui operator de a masca implementari diferite in spatele unei interfete unice Polimorfism (2)

PAGINA 38

O singura paradigma Un singur limbaj utilizat de utilizatori, analisti, proiectanti, etc. Faciliteaza re-utilizarea codului si structurile arhitecturale Modeleaza mai corect lumea reala: Descriere mai precisa a fluxurilor de date si a proceselor aferente Descompuneri naturale ale structurilor mari Usor de inteles si de intretinut Stabilitate O mica schimbare la nivelul cerintelor nu conduce la modificari profunde in sistemul aflat in faza de dezvoltare Puncte tari ale POO

PAGINA 39

Lungimea programelor dezvoltate folosind limbajul OOP este mult mai mare decât abordarea procedurală. Deoarece programul devine mai mare, este nevoie de mai mult timp pentru a fi executat, ceea ce duce la o execuție mai lentă a programului. Nu se poate aplica OOP peste tot, deoarece nu este un limbaj universal. Se aplică numai atunci când este necesar. Nu este potrivit pentru toate tipurile de probleme. Procesul de gândire implicat în programarea orientată pe obiecte poate să nu fie natural pentru unii oameni. Puncte slabe ale POO

CURS 3

PAGINA 2

Analiza, proiectarea si programarea orientata obiect 3 Sistem informatic Acea parte a sistemului informational a unei organizatii in care stocarea / prelucrarea datelor se realizeaza prin mijloace informatice El este format din aplicatia / produsul informatic si elementele folosite pentru exploatarea sa de catre utilizatori Ciclul de viata al SI: defineste perioada de timp dintre “nasterea” (punerea in functiune) si “moarte” (inlocuire) 4 Dezvoltarea aplicatiilor OO 5 Analiza: Intelegerea, gasirea si descrierea conceptelor in domeniul problemei Proiectarea: Structurarea solutiei soft pornind de la etapa de analiza pana la nivel de implementare Analiza si proiectarea OO 6 Analiza vs. proiectare Analiza Evolutie Structura sistem Cerinte functionale Model (in mic) Proiectare Operatii si atribute Performanta Usor de programat Ciclul de viata al obiectelor Cerinte non-functionale Model (in mare) 7 Ciclul de viata al soft-ului Unde se regaseste abordarea OO ? 8 Obiecte vs. functii 9 Cerintele dezvoltarii SI Functionale Nefunctionale: Securitate / protectie Flexibilitate Robustete 10 Modele de dezvoltare (1) Linear Cascada Agile Spirala V-model 11 Modele de dezvoltare (2) Modelul linear 1. Proiectele reale urmează rareori fluxul secvenţial pe care îl propune modelul. 2. Este adesea dificil pentru client să precizeze toate cerințele în mod explicit. Modelul liniar are dificultăţi în adaptarea incertitudinii naturale care există la începutul multor proiecte. 3. Clientul trebuie să aibă răbdare. O versiune de lucru a programului (programelor) nu va fi disponibilă până târziu, pe parcursul implementarii proiectului. O gafă majoră, dacă nu este detectată până când programul de lucru este revizuit, poate fi dezastruoasă. 12 Modele de dezvoltare (3) Modelul linear – dezavantaje: Controlul se modifica pe parcursul procesului Necesita ca toate constrangerile sa fie corect specificate inca de la inceputul procesului Versiunea functionala a unui program este livrata la sfasitul ciclului; erori in faze de inceput pot fi catastrofale. 13 Modele de dezvoltare (4) Modelul cascada ilustreaza procesul de dezvoltare software intr’un flux linear / secvential. Fiecare faza trebuie finalizata inainte ca urmatoarea faza sa poata incepe si nu exista suprapuneri intre faze. Toate cerintele posibile sunt inregistrate in documentele privind cerintele produsului. Fiecare faza corespunde unei activitati si trebuie sa se termine la o anumita data prin producerea anumitor documente sau programe. 14 Modele de dezvoltare (5) Cand se foloseste?  Cerințele sunt foarte bine documentate, clare si fixe;  Definiția produsului este stabilă;  Tehnologia este înțeleasă și nu este dinamică;  Nu există cerințe ambigue;  Aplicația nu este complicata  Când există un flux accentuat al personalului firmei;  Resurse suficiente. 15 Modele de dezvoltare (6) Modelul Agile se concentreaza pe adaptabilitatea procesului și satisfactia clientilor prin livrarea rapida a produselor software funcționale. Metodele Agile impart produsul in mici versiuni incrementale. Aceste versiuni sunt furnizate in iteratii. 16 Modele de dezvoltare (7) Agile vs model cascada abc1 Slide 16 abc1 abc, 10/14/2022 17 Modele de dezvoltare (8) Un „model spiralat” este un tip de model iterativ în care iterațiile individuale din modelul spiralat iau forma unor mini-cascade. Modelul în spirală oferă suport pentru gestionarea riscurilor. Un risc este orice situație adversă care ar putea afecta realizarea cu succes a unui proiect. Cea mai importantă caracteristică a modelului spiralat o reprezinta gestionarea acestor riscuri necunoscute după începerea proiectului. 18 Modele de dezvoltare (9) Cand se foloseste?  Atunci când există o constrângere bugetară și evaluarea riscurilor este importantă.  Pentru proiecte cu risc mediu spre mare.  Angajamentul pe termen lung al proiectului din cauza potențialelor modificări ale priorităților economice pe măsură ce cerințele se schimbă în timp.  Clientul nu este sigur de cerințele sale, ceea ce este de obicei cazul.  Cerințele sunt complexe și necesită evaluare pentru a obține claritate.  O nouă linie de produse ar trebui lansată în etape pentru a obține suficient feedback de la clienti.  Se așteaptă modificări semnificative ale produsului în timpul ciclului de dezvoltare. 19 Modele de dezvoltare (10) V-Model reprezinta un ciclu de viata de dezvoltare software care subliniază conceptul de „verificare și validare”. In fiecare etapă de dezvoltare a modelului V, va exista o faza de testare corespunzătoare care va valida un astfel de proces. Fazele de testare vor fi planificate în paralel cu dezvoltarea etapei pe care se presupune ca vor fi testate De asemenea, este considerat a fi o formă extinsă a modelului de cascadă, deoarece un pas nu poate fi făcut fără finalizarea unui proces anterior. 20 Modele de dezvoltare (11) 21 Scopul trebuie cât mai bine definit și detaliat astfel încât echipa să nu devieze de la acesta. Timpul reprezintă resursa cea mai importantă, deoarece aceasta nu poate fi înlocuită. Orice deviere de la scop aduce timp irosit, iar acest lucru trebuie să fie acoperit în cadrul managementului de risc. Cele 5 dimensiuni: scop, calitate, risc, timp, cost (1) 22 Calitatea reprezinta ceea ce clientul sau stakeholderii au nevoie astfel încât proiectul să fie considerat un succes. La bază, calitatea proiectului este reprezentată de posibilitatea utilizatorului de a folosi acel produs. Managementul riscului reprezintă un proces continuu, proactiv şi sistematic de identificare, evaluare şi gestionare a riscurilor, realizat la nivelul fiecărei entităţi cu scopul de a furniza asigurări rezonabile în ceea ce privește atingerea obiectivelor. Costul este o componentă primordială deoarece scopul principal al proiectului este generarea de profit. Cele 5 dimensiuni: scop, calitate, risc, timp, cost (2) 23 Etapele dezvoltarii soft-ului (1) 1. Etapa de analiza porneste de la specificarea problemei de rezolvat si are ca rezultat modelarea OO: Modelul obiectual: pune in evidenta aspectele statice ale aplicatiei; sunt identificate conceptele cu care se lucreaza si li se asociaza clase si obiecte si relatiile aferente  diagrama de clase si diagrama de obiecte Modelul dinamic: evidentiaza aspectele dinamice ale claselor / obiectelor care au un comportament dinamic semnificativ; sunt stabilite starile prin care poate trece un obiect, evenimentele ce determina tranzitia dintr’o stare in alta, etc. 24 Modelul functional: descrie functionarea, i.e. propietatile comportamentale ale obiectelor; sunt ilustrate relatiile dintre parametrii de I/O ale fiecarei fct, fara detalii privind implementarea algoritmului intr’un limbaj de programare 2. Etapa de proiectare trece de la aspectele conceptuale la elementele legate de implementarea directa a produsului soft Proiectarea sistemului: se iau decizii cu privire la arhitectura aplicatiei (resurse, platforma gazda) si modalitati de implementare (hard / soft) Etapele dezvoltarii soft-ului (2) 25 Etapele dezvoltarii soft-ului (3) Proiectarea obiectelor: descrierea amanuntita a fiecarei clase. Pasi: Structurarea diagramelor de interactiuni pentru fiecare scenariu Structurarea detaliata a diagramelor de clase Etapa de implementare implica transpunerea proiectului intr’un limbaj de programare Etapa de testare: Testarea modulelor individuale Testarea sistemului (ca un tot unitar) 26 Etapele dezvoltarii soft-ului (4) 27 Elaborarea Modelului Obiectual – MO (1) Pentru construirea MO se identifica intai clasele si asociatiile; se adauga atributele si se utlizeaza mostenirea Etape: Se identifica clasele / obiectele; reguli: Evitarea claselor redundante Eliminarea claselor irelevante Declararea ca atribute a proprietatilor celor mai relevante ale obiectelor Se includ in clase operatiile ce sunt aplicate obiectelor Pregatirea unui dictionar de date, i.e. descrierea precisa a fiecarui obiect, asociatii, atribut si operatii 28 Elaborarea Modelului Obiectual – MO (2) Identificarea relatiilor de asociere (orice dependenta intre doua / mai multe clase sau o referire de la o clasa la alta); reguli: Se elimina asocierile irelevante Se elimina asocierile redundante Daca in procesul de analiza se elimina o clasa, asocierile corespunzatoare se elimina / se reformuleaza in functie de alte clase Majoritatea asocierilor intre 3 / mai multe clase pot fi descompuse in asocieri binare Indentificarea atributelor/operatiilor Organizarea claselor prin mostenire: De jos in sus: generalizare in superclasa De sus in jos: prin rafinarea claselor existente 29 Iterarea modelului Gruparea claselor in module (o multime de clase ce reprezinta o submultime logica a intregului model) Definitii: Diagrama de clase: reflecta instatieri posibile ale datelor Diagrama de obiecte: descrie modul in care interrelationeazaun set de obiecte Relatia de agregare: daca asociatia exprima o relatie de tip “parte–intreg” Elaborarea Modelului Obiectual – MO (3) 30 Are la baza 2 concepte: Stare: Caracterizeaza setul de valori asociate obiectului Este o abstractizare a valorilor atributelor si legaturilor unui obiect Reprezinta raspunsul unui obiect la un eveniment (extern) Corespunde intervalului dintre 2 evenimente Elaborarea Modelului Dinamic – MD (1) 31 Eveniment: Caracterizeaza stimulii externi ce actioneaza asupra obiectelor. Reprezinta o transmitere unidirectionala de informatie (de la un obiect la altul). Special: erori Concurenta: doua sau mai multe evenimente necorelate cauzal Scenariu: secventa de evenimente ce apare intr’un caz particular de evolutie a sistemului. Elaborarea Modelului Dinamic – MD (2) 32 Elaborarea Modelului Dinamic – MD (3) Operatii: Activitati: Sunt asociate starilor si necesita timp pentru a fi efectuate. Actiuni: Operatii instantanee asociate evenimentelor. MD al unei clase este mostenit de subclasele sale; acestea mostenesc atat starile cat si tranzitiile 33 Descrie modul in care se efectueaza calculele in interiorul unui sistem, fara a tine cont de secvente, decizii sau structuri Proces: transforma valorile datelor; rezultatele unui proces depind de comportamentul sistemului, specificat prin MD Flux de date: conecteaza iesirea unui proces cu intrarea altui proces Elaborarea Modelului Functional – MF 34 Ciclul de dezvoltare software

1 M. Caramihai, © 2022 PROGRAMAREA ORIENTATA OBIECT 2 CURS 4

Introducere in UML (1) 3 Ce este modelarea ? Aspecte generale privind modelarea Trasaturile modelarii: Simplificare (rezultatul abstractizarii) Subordonare unui scop Reprezentarea unei realitati Divizare Ierarhizare Comunicare Grupuri tinta: client/utilizatori si membrii echipa proiect 4 Ce este modelarea vizuala (1) ? Modelarea vizuala permite modelarea prin notatii grafice standard Principii: Modul de creare tine de scopul utilizarii modelului Orice model poate fi conceput la diferite niv de abstractizare Orice sistem real poate fi reprezentat printr’o suita de modele 5 Ce este modelarea vizuala (2) ? Prin modelarea vizuala se pot abstractiza obiectele & logica proceselor de afaceri Prin modelarea vizuala poata fi analizata & proiectata o aplicatie (informatica) 6 UML este prescurtarea de la Unified Modeling Language Definitie: limbaj de modelare pentru specificarea, vizuualizarea, constructia si documentarea componentelor unei aplicatii informatice UML reprezinta o sinteza a: Conceptele Data Modeling (Entity Relationship Diagrams) Business Modeling (work flow) Modelarea obiectelor Modelarea componentelor UML este limbajul standard “for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system” [The UML definition was led by Grady Booch, Ivar Jacobson, and Jim Rumbaugh ], http://www.omg.org Focus pe elementele conceptuale si fizice ale reprezentarii unui sistem. Ce este UML ? 7 Istoria dezvoltarii UML 8 Conceptele UML UML poate fi utilizat pentru: Descrierea limitelor unui sistem si a functiilor sale principale (use cases & actori)  comportamentul functional al sistemului Ilustrarea implementarii cazurilor de utilizare prin diagramele de interactiuni Reprezentarea structurii statice a unui sistem prin diagrama de clase (utilizarea claselor+interfetelor pt model. entitatilor statice din sistem) si diagrama de obiecte (arata instantele claselor + legaturi) Modelarea evolutiei obiectelor prin diagrama de tranzitii Structurarea implementarii fizice a arhitecturii sistemului cu diagramele de componente si operationale  arata org. elementelor din sistem Diagrama de pachete: tip special de diagrame de clase; focus pe gruparea claselor Diagrama de activitati: descr flux de evolutie ale activitatilor Diagrama d secvente: surprinde tipul si ordinea mesajelor 9 Arhitectura UML (1) Arhitectura = setul de decizii privind: i. Organizarea sistemului software. ii. Selectia elementelor structurale & interfete din care un sistem este format. iii. Evolutia & colaborarea elementelor. iv. Structura si evolutia elementelor. v. Arhitectura sistemului. 10 Arhitectura UML (2) Use Case View Analiza Cazurilor de utilizare (CU) este o tehnica de “captura” a proceselor din perspectiva utilizatorului. Include evolutia “vazuta” de utilizatori, analisti si testatori. Specifica “fortele” ce influenteaza arhitectura. Aspectele statice sunt reprezentate in diagramele CU. Aspectele dinamice sunt reprezentate in diagramele de interactiuni, de activitati si de stari Design View Cuprinde clasele, interfetele si colaborarile ce definesc “vocabularul” unui sistem. Definesc necesitatile functionale ale sistemului. Aspectele statice sunt reprezentate in diagrama de clase si diagrama de obiecte 11 Arhitectura UML (3) Process View Defineste cozile si procesele concurentiale si de sincronizare. Reprezinta performanta si scalabilitatea. Aspecte statice si dinamice reprezentate la fel ca in Design View. Implementation View Defineste componentele si fisierele utilizate in realizarea unui sistem fizic. Reprezinta managementul configuratiei. Aspectele statice sunt reprezentate in diagrama de componente. Aspectele dinamice sunt redate in diagramele de interactiuni, de activitati si de stari. Deployment View Defineste nodurile ce formeaza topologia hardware. Reprezinta distributia si instalarea sistemului. Aspectele statice sunt reprezentate in diagrama de distributii. Aspectele dinamice sunt redate in diagramele de interactiuni, de activitati si de stari. 12 Arhitectura UML (4) Arhitectura structurata Meta-metamodel: limbaj specific metamodelului Metamodel: limbaj specific modelului Model: limbaj specific domeniului Obiecte: instante ale unui model 13 Arhitectura UML (5) Model sistemic Imagine de ansamblu = suma diagramelor UML 14 Metamodelul UML (1) 15 Metamodelul UML (2) 16 Metamodelul UML (3) 17 Metamodelul UML (4) 18 Elemente constructive in UML Lucruri Concept de modelare (elementul de individualitate). Relatii Leaga elementele individuale (I.e. conceptele) Diagrame Grupeaza colectiile organizate de elementele individuale (lucruri / relatii) 19 Lucruri - componente Structuri: “substantivul” modelelor UML Evolutii: componenta dinamica a modelelor UML Grupari: componenta organizationala a modelelor UML Adnotari: componenta explicativa a modelelor UML 20 Elemente conceptuale sau fizice Structuri 21 Clase (1) Din punct de vedere grafic, o clasa este redata ca un dreptunghi; Include in mod uzual numele clasei, atribute si operatii (in compartimente separate). Numele clasei este obligatoriu Atributul este numele propietatii unei clase; forma: attributeName: Type 22 Clase (2) Un atribut derivat poate fi calculat din alte atribute (el ne-existand in forma directa) d.e. varsta unei persoane poate fi calculata in raport de data nasterii si data curenta; reprezentare: / varsta: Date 23 Clase (3) Atributele pot fi: + public (permit accesul la date si metode din afara clasei) # protected (interzic accesul din afara clasei, dar il permit din clasele derivate) - private (interzic accesul la date si metode in afara clasei) / derived 24 Clase (4) Vizibilitatea membrilor unei clase si reprezentarea in UML 25 Clase (4) Operatiile: descriu evolutia unei clase 26 Descrie un set de operatii ce specifica evolutia obiectelor fara a descrie structura lor interna. Se reprezinta cu stereotipul <> in fata numelui Interfata Interfata nu poate fi instantiata Nu au atribute sau stari Pot specifica serviciile oferite de o clasa asociata 27 Vizibilitatea claselor in C++ 28 Relatii (1) Relatiile ofera o structura de comunicare intre obiecte Diagramele de secvente / colaborari sunt examinate pentru a determina ce legaturi trebuiesc stabilite intre obiecte pentru a permite evolutia sistemului; daca doua obiecte “vorbesc”, o legatura (link) trebuie sa existe intre ele. Tipuri de relatii: Dependenta – o clasa utilizeaza o alta clasa (“uses”) Asociere – o clasa este in relatie cu o alta clasa pe o durata mai mare de timp (“has a”) Generalizare Realizare – unul dintre elementele relatiei garanteaza finalizarea asteptatat din partea celuilalt element 29 Relatii (2) 30 Relatia de dependenta indica o relatie semantica intre doua sau mai multe elemente. O schimbare intr’unul din obiecte (independent, sursa) conduce la modificari semantice la celalalt obiect (dependent, destinatie) Dependenta (1) Orar Add(c:Course) Remove(c:Course) Curs 31 Caracteristici: Dependența semnifica „O clasă o folosește pe cealaltă” O relație de dependență indică faptul că o schimbare într-o clasă poate afecta clasa dependentă, dar nu neapărat si invers. O relație de dependență este adesea folosită pentru a arăta că o metodă are obiectul unei clase ca argument. Forme predefinite: Refine: “rafinarea” unui element de modelare prin intermediul altuia Trace: acelasi concept, dar pe un nivel de abstractizare diferit Use: relatia prin intermediul careia un element solicita prezenta unui alt element pentru buna sa desfasurare. Dependenta (2) 32 Dependenta – implementare in C++ class A { ... }; class B {public:void Executa1(void);}; void B::Executa1(void) {A a; a \* a1 = new A; // ... delete a1;} B Executa A 33 Generalizare (1) O relatie de generalizare / specializare leaga o subclasa de o superclasa. Indica o mostenire a atributelor si actiunilor de la nivelul superclasei la cel al subclasei (sau o specializare la nivelul subclasei a elementelor generale din superclasa) Reprezentare Persoana Student 34 Generalizare (2) UML permite ca o clasa sa mosteneasca mai multe superclase. Observatie: anumite limbaje OO (d.e. Java) nu permit mostenirea multipla Doctorand Angajat Asistent 35 Generalizare (3) Mostenire multipla Avantaje: • Structurarea elementelor in diferite forme si cu diferite legaturi (ca in lumea reala) • Posibilitati multiplede utilizare a atributelor si operatiilor din clasele parinte Dezavantaje • Orice modificare la nivelul superclasei duce la modificari si in subclasa • Cand o subclasa mosteneste aceleasi attribute / operatii de la o superclasa, trebuie selectat cu grija ce va fi utilizat 36 Generalizare (4) Restrictii: Overlapping: un element apartine simultan la mai multe subclase Disjoint: contrar situatiei anterioare Complete: sunt specificate toate subclasele Incomplete: alte subclase pot sa apara ulterior 37 Generalizare – implementare in C++ class Persoana{ ... }; class Student : public Persoana{ ... }; Persoana Student 38 Asociere Daca doua clase (in cadrul unui model) trebuie sa comunice una cu alta, trebuie sa existe o legatura (link) intre ele. Asocierea (delegarea) reprezinta o asemenea legatura. Reprezentare: sau Student FACULTATE Student FACULTATE invata la 39 Proprietatile asocierii Nume Numele asocierii Rol (nu mai exista in UML2; inlocuit cu “association end name”) Rolul specific al asocierii Multiplicitate Indica numarul de obiecte ce sunt conectate Tip Asociere, agregare, compozitie; Caz particular: asociere reflexiva (in obj aceleiasi clase) Directie Calificator: atribut / grup de atribute a caror valoare serveste pt partajarea ansamblului de obj. participante la o asociere 40 Asocierea - exemplu Profesor Curs \* 1..\* preda Nume relatie Multiplicitate (defineste numarul de obiecte asociate cu o instanta a asocierii Indicator de directie (cum se citeste numele relatiei) 41 Multiplicitatea O clasa poate fi legata de alta printr’o relatie:  One-to-one  One-to-many  One-to-one or more  One-to-zero or one  One-to-a bounded interval (one-to-two through twenty)  One-to-exactly n  One-to-a set of choices (one-to-five or eight) Exprimarea multiplicitatii:  Exact unu -1  Zero sau unu - 0..1  Multe - 0..\* sau \*  Unu sau mai multe - 1..\*  Valoare exacta - d.e. 3..4 sau 6  Reprezentare complexa –d.e. 0..1, 3..4,6..\* semnifica orice numar de obiecte, altul decat 2 sau 5 42 Asocierea reflexiva Asocierea a 2 instante ale aceleiasi clase O clasa are responsabilitati multiple:  d.e. angajatul unei firme poate fi seful unui grup format din 10 alti angajati. Angajat salariat manager coordoneaza 43 Multiplicitatea asocierilor (1) Multi-la-unu: o banca are mai multe ATM-uri; 1 ATM este legat doar la o banca Unu-la-multi: un inventator are mai multe inventii, o inventie este a unui singur inventator ATM Banca \* 1 Inventator Inventie 1 \* 44 Multiplicitatea asocierilor (2) O echipa de fotbal are 11 jucatori. Unul dintre ei este capitanul echipei. Un jucator poate juca numai intr’o echipa. Un capitan de echipa poate conduce o singura echipa. Jucator capitan membru al echipei conduce Echipa 10 1 11 1 1 0..1 membru al capitan 45 Asocieri duale Jucator Echipa 1 1..\* liderul unei component al 1..\* 1..\* 46 Clasa unei asocieri Asocierile pot fi si obiecte in sine si reprezinta instantieri ale unor clase (link classes / association classes). Produs Garantie Certificat Model Serie\_produs Numar\_certificat 47 Asociere – agregare - compunere Agregarea defineste o relatie “parte– intreg”, in care “partea” poate exista in afara “intregului”. Este introdusa o relatie de tip: "has a". Compunerea este un tip special de agregare: intregul contine mai multe parti, iar partile nu pot exista in afara intregului. Elementele componente “traiesc” si “mor” odata cu intregul Facultate Student Corp uman Inima 48 Agregare – reprezentare in C++ class B { ... }; class A { private: A \* a; }; Observatie: a este instantierea lui A A B 49 Compozitie – reprezentare in C++ class B { ... }; class A { private: A a; }; Observatie: a este instantierea lui A A B 50 Exemple School Department 1 1..\* Student 1..\* \* 51 O relatie semantica intre doua elemente, in care unul dintre elemente “garanteaza” finalizarea actiunilor asteptate din partea celuilalt element. Realizare Interfata: specifica operatiile vizibile ale unei clase / pachet fara a define si structura interna a acestora Realizarea este relatia ce leaga o clasa de interfata aferenta Observatie: in UML interfata este o clasa abstracta Realizare <> ControlPanel FunctiiSistem Implementeaza specifica 52 Exemple Faculty & student (asociere) Hospital & doctor (asociere) Door & Car (agregare) Member & Organization (agregare) People & student (generalizare) Circle & point (generalizare) Department & Faculty (asociere) Employee & Faculty (generalizare) Item & Printer (generalizare) 53 class A { public: virtual void Executa1(void) = 0; virtual int Executa2(void) = 0; }; class B : public A { public: virtual void Executa1(void); virtual int Executa2(void); }; void Bar::doSomething(void) {...}; Int Bar::doSomethingOther(void) {...}; Interfata – implementare in C++ <> A Executa1=0 Executa2:int=0 B Nu este realizabila in aceasta forma. Se poate implementa o clasa virtuala (fara atribute, fara metode private / protejate, fara implementari de metode) 54 Reguli generale Modelarea relatiilor Se utilizeaza dependentele daca relatiile nu sunt structurate (in alt mod). Se utilizeaza generalizarea in cazul unei relatii “is-a”. Nu se recomanda introducerea generalizarilor ciclice. Balance generalizations - Not too deep, not too wide. Trasarea unei relatii in UML Se vor folosi in special liniile drepte si cele oblice. Se va evita intersectia liniilor. Vor fi reprezentate doar relatiile necesare pentru o buna intelegere a unui grup de clase / obiecte. Se vor evita asocierile redundante.

1 M. Caramihai, © 2022 PROGRAMAREA ORIENTATA OBIECT 2 CURS 5

Introducere in UML (2) 3 “A use case specifies the behavior of a system or a part of a system, and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.” -The UML User Guide, [Booch,99] “An actor is an idealization of an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that outside users may have with the system.” -The UML Reference Manual, [Rumbaugh,99] Cazuri de utilizare (CU) 4 Ce este un CU ? Un caz de utilizare este: Un set de scenarii ce descrie evolutia unui sistem impreuna cu utilizatorii sai Descrie functionalitatea pe care un sistem o pune la dispozitia utilizatorilor sai (oameni / sisteme) si a legaturilor dintre ei. Defineste necesitatile clientilor Actor: oameni / sisteme ce se gasesc in afara sistemului (interactioneaza cu sistemul) Scenariu: dialog intre actor si sistem 5 Scopul CU Descrierea cerintelor functionale ale sistemului Structurarea unui cadru de referinte comun Formarea bazei de testare si verificare Reprezentarea interactiunilor sistem-utilizator  Totalitatea CU: functionalitatea complete a sistemului Observatie: CU este initiat de catre un actor si descrie o suita de interactiuni dintre acesta si entitatea (informatica), interactiuni reprezentate printr’un schimb de mesaje 6 Identificarea CU Identificarea CU și apoi procesul de evolutie bazat pe scenarii se bazeaza pe intrebarea: ce valoare vizibilă externă și observabilă își dorește fiecare actor, de ex: Ce funcții își va dori actorul de la sistem ? Sistemul stochează informații? Ce actori vor crea, vor citi, vor actualiza sau vor șterge aceste informații? Este necesar ca sistemul să notifice un actor despre schimbările din starea internă? Există evenimente externe despre care trebuie să știe sistemul? Ce actor informează sistemul despre aceste evenimente? 7 Relatii intre CU Asociere: participarea unui actor la un CU Extensie: un CU poate fi extins cu un comportament definit de un alt CU Indică faptul că un caz de utilizare „Invalid password” poate include (sub rezerva specificarii în extensie) comportamentul specificat de cazul de utilizare de bază „Login account”. 8 Relatii intre CU Incluziune: o instanta a unui CU cuprinde si comportamentul specificat printr’un alt CU Când un caz de utilizare este descris ca utilizând funcționalitatea altui caz de utilizare, relația dintre cazurile de utilizare este denumită ca include sau use. Generalizare: Cazul de utilizare copil reprezinta o îmbunătățire a cazului de utilizare părinte. 9 CU - exemplu Actorii sunt conectati la cazurile de utilizare numai prin relatii de asociere. 10 Diagramele UML 11 Diagrama de secvente (1) 12 Diagrama de secvente (2) Diagrama de secvente este o diagrama de interactiuni care reda ordonarea in timp a mesajelor. Ea prezinta un set de obiecte si mesajele trimise si primite de acele obiecte. Din punct de vedere grafic, diagrama de secvente este un tabel ce prezinta obiectele pe orizontala si mesajele (ordonate in sensul parcuregerii timpului) pe verticala. Continut Obiecte: Schimba mesaje unul cu altul Mesaje: Sincrone: • Uzual: apel operație – se trimite mesajul și si se suspenda execuția în așteptarea răspunsului • reprezentate prin “sageata plina”; durata trebuie sa fie indicata prin bara de activare sau sageata de intoarcere 13 Diagrama de secvente (3) Asincrone: • Se trimite mesajul și si se continua imediat fără a aștepta valoarea returnată • reprezentata prin “jumatate de sageata” Pot exista mesaje de tip «create» si«destroy» Tipuri speciale Lost message: mesajul nu ajunge niciodata la destinatie (d.e. ping) Found message: mesajul a carui origine nu este cunoscuta (d.e. exception handling) 14 Reprezentare Un obiect este redat ca o cutie din care coboara o linie intrerupta. Linia poarta numele de linia de viata a obiectului (reprezinta durata de viata a unui obiect pe un anumit interval de timp). Mesajele sunt redate cu sageti orizontale (“coboara” odata cu trecerea timpului) Conditiile ( d.e. [check = “true”] ) indica faptul ca un mesaj a fost transmis. Ordinea obiectelor nu este semnificativa Un indicator de iteratie (d.e. \* sau [i = 1..n] , indica faptul ca un mesaj va fi repetat de mai multe ori (corespunzator valorii precizate) 15 Diagrama de secvente - observatii Diagramade secvente reprezinta evolutia sistemului in raport cu interactiunile. Reprezinta o complementaritate a diagramei de clase (care reprezinta structura sistemului informatic) Foarte utila in identificarea obiectelor implicate in diverse activitati. Buget mare de timp pentru implementare. Principiul de realizare: KISS (Keep It Small & Simple) 16 Diagrama de activitati (1) 17 Diagrama de activitati (2) Diagrama de activitati (DA) este un flowchart si indica fluxul de control de la o activitate la alta. Diagrama de activitati specifica, dezvolta si documenteaza dinamica unei “societati de obiecte” Diagrama de interactiuni descrie fluxul de control de la un obiect la altul; diagrama de activitati descrie fluxul de control de la o activitate la alta. Exista doua feluri de stari: Actiuni (Action state): • Nu pot fi descompuse • Sunt “instantanee” (in raport cu nivelul de abstractizare din model) Activitati (Activity state): • Pot fi descompuse • Activitatea este modelata de o alta diagrama de activitati 18 Diagrama de activitati (2) Utilizari: Identificare cazuri de utilizare candidate, prin examinarea fluxurilor de lucru în afaceri Identificare condițiile pre- și post (contextual) pentru cazurile de utilizare Modelare fluxuri de lucru între/în cadrul cazurilor de utilizare Modelare fluxuri de lucru complexe în operațiuni pe obiecte Modelare în detaliu activități complexe într-o activitate de nivel înalt 19 Caracteristici Seamana foarte mult cu diagramele clasice Identifica activitatile din cadrul unui sistem (si se bazeaza pe cazurile de utilizare) Identifica tranzitiile intre activitati Descrie comportamentul unei clase ca raspuns la calculele interne. Orientata mai mult catre business process decat catre OO Foarte utila in faza de testare Diagrama de activitati descrie fluxul din interiorul unui system Diagrama de activitati este un caz special al diagramei de stari in care starile sunt inlocuite cu activitati (functii). 20 Notatii 21 Exemplu © www.developer.com 22 Diagrama de colaborari (1) 23 Diagrama de colaborari (2) Prezinta modul in care obiectele interactioneaza unele cu altele (tinand cont de unitatile organizationale) Secventa de mesaje este determinata prin numere: 1, 2, 3, 4, ….. 1, 1.1, 1.2, 1.3, 2, 2.1, 2.1.1, 2.2, 3 (prezinta modul in care o operatie apeleaza o alta operatie) Observatie: DC – Diagrama de comunicare (in UML2) Continut Obiecte • Schimba mesaje unele cu altele Mesaje • Sincrone: reprezentate prin sageata completa • Asincrone: “semnale” reprezentate prin sageata incompleta • Mesaje de tip «create» sau «destroy» • Mesajele sunt numerotate si pot avea “bucle” 24 Exemple © www.dotnetcoders.com 25 Comparatie DC / DS Reprezentarea diagramei de colaborari prezinta conexiunea dintre obiecte. Diagrama de secvente permite o buna reprezentare a fluxului de timp Secventa de mesaje este mai greu de inteles intr’o diagrama de colaborari Organizarea obiectelor (si fluxul de control) sunt cel mai bine identificate in diagrama de colaborari Observatii: Controlul complex este greu de reprezentat printr’o singura diagrama!!! Diagrama de colaborari este mai des folosita decat cea de secvente 26 Sumar 1. Diagrama CU [Model Functional] modeleaza functionalitatea dpdv user 2. Diagrama de clase [Model Obiectual] modeleaza structura sisemului utilizand obiecte 3. Diagrama de interactiuni [Model Dinamic] (secvente & colaborari) modeleaza mesajele schimbate intre obiecte 4. Diagrama de stari [Model Dinamic] modeleaza tranzitiile intre stari 5. Diagrama de activitati [Model Dinamic] modeleaza fluxul de control (ca tranzitie intre activitati) 27 Sisteme de dimensiuni mari Principiul Roman: Divide & impera Sistemele mari trebuiesc desfacute in componente mai mici, pentru a putea fi mai usor gestionate Metode structurale: descompunere functionala In OO: gruparea claselor in unitati cu caracteristici comune. Pachete (Packages) (conceptual: in momentul dezvoltarii sistemului) Componente (fizic: in momentul rularii) Diagrama de pachete poate fi folosită pentru a simplifica diagramele de clase complexe (i.e. poate grupa clase în pachete) Un pachet reprezinta o colecție de elemente UML legate logic. 28 Pachete (Packages) Un pachet reprezinta o “grupare” de elemente de modelare; poate contine clase sau alte pachete / diagrame 29 Elemente Nume (string) Simple Calificator: numele P are ca prefix numele P in care se gaseste, d.e. Pachet1::Pachet2 Componenta: un P poate contine diferite elemente (d.e. clase interfete, diagrame, alte pachete, etc) Observatii: • daca un P este distrus, sunt distruse si toate componentele sale • Orice alement apartine unui singur P • P se recomanda a fi pe max 2-3 nivele Vizibilitate: idem ca la clase (usual: public) 30 Pachete - exemple Vanzari Client Ordin Depozit Locatie Item Stoc Ordin 31 Pachete – caracterizare Un pachet poate contine diferite tipuri de elemente de modelare Pot fi incluse si alte pachete in vederea dezvoltarii de ierarhii. Un pachet defineste un “spatiu” pentru elementele ce le contine (similar cu directoarele din DOS / Windows) Pachetele sunt utilizate in general ca structuri de grupare a unor elemente cu o semantica comuna. Diagrama prezinta o “vedere de sus” asupra sistemului. 32 Concepte de baza Package Grup de elemente de modelare. Import Relatie de dependenta; indica faptul ca elementele de continut ale pachetului tinta sunt adaugate spatiului pachetului sursa (extinde spatial de lucru al P importator) Access Relatie de dependenta; indica faptul ca elementele de continut ale pachetului tinta pot fi accesate in raport cu spatiul pachetului sursa (permite utilizarea elementelor dintr’un alt P prin specificarea caii de acces la acestea). Nume <> <> 33 Concepte de baza Package Import Un pachet importa functionalitatile celuilalt pachet Access Un pachet necesita un “ajutor” din partea celuilalt pachet pentru a functiona 34 Relatii intre pachete – exemplu 35 Mostenirea Un pachet aflat in relatie de generalizare cu un alt pachet va mosteni elementele publice / protejate ale acestuia daca: Elementele apartin pachetului de la nivelul ierarhic superior Elementele sunt importate de catre pachetul de la nivelul ierarhic superior 36 Reprezentarea pachetelor (continut ) Pachetele sunt reprezentate prin intermediul diagramelor statice Exista doua modalitati echivalente de reprezentare: 37 Utilizarea pachetelor Cerinte Grd cat mai inalt de coeziune interna Nivel cat mai redus de cuplare externa Unitatea scopului Utilizari Crearea unei “vederi de ansamblu” asupra unui mare numar de elemente de modelare Organizarea modelelor de mari dimensiuni Gruparea elementelor interconectate Separarea spatiilor de lucru 38 Pachete – reguli de configurare Elementele de modelare puternic cuplate trebuie sa se gaseasca in acelasi pachet Elementele de modelare slab cuplate trebuie sa se gaseasca in pachete diferite. Trebuisc evitate relatiile (in special asocierile) dintre elementele de modelare aflate in pachete diferite. Un element importat intr’un pachet nu trebuie sa “cunoasca” cum este folosit in acel pachet. 39 Sisteme & subsisteme Sisteme: un set e elemente de organizare utilizate in vederea implementarii unui scop Legatura dintre sisteme si subsisteme este de tip compozitie. Arhitecturi C/S Structuri monolitice Pe 2 nivele Pe 3 nivele 40 Subsisteme Subsistemele sunt utilizate in descompunerea sistemelor. Grup de elemente de modelare ce formeaza impreuna o unitate de comportament in sistem fizic 41 Subsistem – exemplu Un sistem poate fi format din subsisteme ce comunica intre ele. Interconectare: prin utilizarea unui subsistem la nivel interfata pentru reducerea interdependentelor SS2 SS3 SS1 42 Concepte de baza Subsistem Un grup de elemente de modelare ce reprezinta o unitate de evolutie intr’un sistem fizic. Subsistemele se definesc dpdv: functional, logic si avand coeziune fizica 43 Subsisteme – componente Un subsistem are doua componente: O componenta externa – prezinta serviciile furnizate de subsistem (i.e. specificare ) O componenta interna – prezinta configurarea subsistemului (i.e. realizare ) Exista o mapare intre cele doua componente. 44 Caracteristicile subsistemelor Un subsistem are elemente de specificare, respectiv de realizare pentru a putea implementa cele doua componente ale sale Operatii Elemente de specificare Elemente de realizare 45 Elemente de realizare Elementele de realizare definesc continutul actual al subsistemului.  Elementele de realizare consta in: •Clase si relatiile dintre ele •Ierarhii de subsisteme 46 Elemente de specificare (1) Elementele de specificare definesc vizibilitatea externa a subsistemului 47 Elementele de specificare: Descriu serviciile oferite de subsistem Descriu evolutia exterioara a subsistemului Nu dau nici o informatie despre structura interna a subsistemului Descriu interfata subsistemului Elemente de specificare (2) 48 Cand se utilizeaza subsistemele ? Descompunerea sistemelor mari in module / componente mai mici Dezvoltarea sistemelor distribuite Asamblarea unui set de module intr’un sistem de dimensiuni mai mari Dezvoltarea sistemelor bazate pe structuri multiple 49 Subsisteme – reguli de configurare Un subsistem se defineste pentru fiece parte a unui sistem de mari dimensiuni Specificatiile tehnice sunt dependente de tipul sistemului / subsistemului Fiece subsistem trebuie sa fie proiectat in mod independent 50 Modele Un model surprinde o “vedere” asupra unui sistem si incearca o “descriere” completa a acestuia In UML 2.5 nu este foarte consistenta definirea domeniului de aplicare a modelului. Într’un loc se spune că un model surprinde un view a unui sistem fizic, în timp ce în altul - acel sistem este înțeles în sensul cel mai larg și poate include nu numai software și hardware, ci și organizații și procese. 51 Concepte de baza Model O viziune asupra unui sistem, intr’un anumit scop, in vederea determinarii aspectelor semnificative ale sistemului Trace O conexiune intre elementele de modelare care caracterizeaza acelasi concept in raport cu mai multe modele. (nu exista o relatie directa intre modele ci doar relatii de tip “trace” intre elementele diferitelor modele) <> 52 Trace Este un stereotip al relatiei de dependenta Nu se aplica elementelor ce apartin aceluiasi model Reprezinta (generic) o dependență ce indică o relație istorică sau de proces între două elemente ce reprezintă același concept fără reguli specifice pentru derivarea unuia din celălalt. 53 Modele vs. diagrame Diagramele specifica detaliile tehnice ale unui model. 54 Utilizarea modelelor Modelele se folosesc pentru: A oferi diferite “viziuni” asupra sistemului pentru categorii diferite de utilizatori A sublinia diversele aspecte ref. evolutia sistemului in timp A evidentia diversele etape de evolutie a unui sistem informatic

1 M. Caramihai, © 2022 PROGRAMAREA ORIENTATA OBIECT 2 CURS 6

Alte limbaje vizuale (UML vs. SysML, BPMN, IDEF) 3 SysML 4 4/15/2008 Modul de descriere a sistemelor Specificatii Interfata Proiectare sistem Analiza & implementare Testare Trecerea de la “modelul documentar” la “modelul sistemic” Inainte In viitor 5 Ce este SysML (1) ? Un limbaj de modelare vizuala – raspuns la UML for Systems Engineering RFP dezvoltat OMG, INCOSE, si AP233. adoptat de OMG in Iunie 2006 Suporta specificarea, analiza, proiectarea, verificarea si validarea sistemelor (in sens general): se include hardware, software, date, personal, proceduri si facilitati Suporta modelarea si schimbul de date prin intermediul standardului XMI ® (XML Metadata Interchange) 6 Este un limbaj de modelare vizuala care ofera: Semantica = intelegerea conceptelor Notatii= reprezentarea conceptelor Nu este o metodologie sau un instrument: SysML este independent de orice limbaj de programare UML – Software Engineering SysML – System Engineering Ce este SysML (2)? 7 SysML - Concepte Modelarea sistemelor trebuie sa permita integrarea tuturor elementelor aferente acestora. SysML (Systems Modeling Language) este un limbaj de modelare al aplicatiilor din ingineria de sistem. Suporta specificarea, analiza, designul, verificarea si validarea unei game vaste de sisteme si sisteme de sisteme. Aceste sisteme pot include hardware, software, informatii, procese si facilitati. Ofera posibilitatea de a unifica concepte inrudite software si non-software, astfel umpland distanta ce se afla intre ele. 8 Copyright © 2006-2008 by Object Management Group. Reprezentarea unui “sistem de sisteme” Limitari Interactiuni 9 Modelarea nivelelor ierarhice