**Cristea Andrei**

**443-A**

Limbajul UML

UML este un limbaj standard pentru specificarea, vizualizarea, construirea si documentarea sistemelor software.

Limbajul a fost creat de către Object Management Group (OMG) care a mai produs printre altele și limbajul de programare CORBA. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT. Așa se face că există aplicații ale UML-ului pentru management de proiecte, pentru business Process Design etc.

Compania OMG incearca sa transforme modelul UML intr-un standard de industrie.

* UML este acronim pentru Unified Modelling Language
* UML este diferit de limbajele comune de programare precum C++, Java, COBOL, etc.
* UML este un limbaj grafic cu ajutorul caruia se creeaza structurile aplicatiilor software.

UML nu este un limbaj de programare dar diferite utilitare pot fi folosite pentru a genera cod pornind de la diagramele UML.

# Istoric

Prima versiune de UML, UML 1.0, a apărut în anul 1990 ca reacție a numeroaselor limbaje de modelare propuse pe piață. UML îi are ca fondatori pe Grady Booch, Ivar Jacobson și James Rumbaugh, așa numiții „cei trei Amigos”. Ei au dezvoltat limbajul bazându-se inclusiv pe limbaje de modelare deja existente, însă incomplete ca gamă de funcționalități. Printre acestea se numără și OOSE, RDD, OMT, OBA, OODA, SOMA, MOSES și OPEN/OML.

# Modelul UML conceptual

* Modelul conceptual este definit ca un model compus din concepte si relatiile dintre acestea
* Inainte de a realiza o diagrama UML, este nevoie de modelul conceptual pentru a identifica entitatile in mod realist si felul in care ele interactioneaza.

UML descrie sisteme reale si de aceea este foarte importanta realizarea unui model conceptual pentru ca apoi sa se continue treptat. Modelul conceptual UML poate fi inteles prin invatarea urmatoarelor trei elemente principale:

* Blocurile UML
* Relatiile dintre blocuri
* Mecanismele uzuale

# Principii orientate pe obiect

UML poate fi vazut drept succesorul analizei si design-ului obiect-orientate.

Un obiect contine atat variabile cat si metode pentru controlul datelor. Variabilele descriu starea obiectului si o clasa descrie un obiect. Clasele pot fi organizate in ierarhii ce se aseamana cu siteme din viata reala. Ierarhia este exprimata prin mostenire si clasele pot fi asociate in diferite modalitati, in functie de cerinte.

Obiectele reprezinta entitati din viata reala si toate conceptele de baza precum abstractizare, encapsulare, mostenire si polimofism, pot fi reprezentate folosind diagrame UML.

* Obiecte: reprezinta entitati si sunt blocurile fundamentale ale diagramelor
* Clase: schema unui obiect
* Abstractizare: reprezinta comportamentul entitatilor din viata reala
* Encapsulare: mecanismul de grupare a datelor si de ascundere a lor de obiecte externe
* Mostenire: mecanismul de a creea clase pornind de la una deja existenta
* Polimorfism: mecanismul prin care obiectele pot exista sub mai multe forme

# Analiza si design OO

Analiza orientata pe obiect reprezinta o investigare a obiectelor. Design-ul inseamna colaborarea dintre obiectele identificate.

Scopul analizei si design-ului OO poate fi descris astfel:

* Identificarea obiectelor dintr-un sistem
* Idenfificarea relatiilor dintre acestea
* Creearea unui design ce poate fi transformat in executabile ce folosesc limbaje orientate pe obiect

# Arhitectura

Orice sistem in viata reala este utilizat de diferite tipuri de utilizatori. Utiizatorii pot fi dezvoltatori, testari, business analisti, etc. De aceea, inainte de a crea un sistem, cea mai importanta parte este de a intelege sistemul din perspectiva diferitor utilizatori.

Uml joaca un rol important in definirea perspectivelor diferite asupra sistemului, cum ar fi:

* Design: cuprinde clase, interfete si colaborari.
* Implementare: defineste gruparea componentele ce realizeaza un sistem fizic complet
* Proces: defineste fluxul sistemului
* Instalare: reprezint nodurile fizice ale sistemului care formeaza hardware-ul

Diagrama care uneste cee patru perspective se numeste Use Case si reprezinta functionalitatea sistemului.

**Angelescu Raluca**

**443-A**

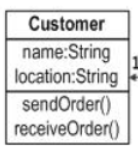
# Descrierea blocurilor UML

Blocurile UML pot fi descrise prin:

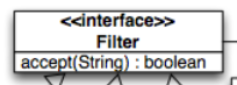
1. Obiectele, cele mai importante blocuri UML ce pot fi:
2. Structurale, ce definesc partea statica a modelului. Reprezinta elementele fizice si conceptuale.

Exemple de obiecte structurale:

* Clasa: reprezinta un set de obiecte ce au roluri similare.



* Interfata: reprezinta un set de operatiuni ce specifica rolurile unei clase.



* Colaborarea: defineste interactiunea dintre elemente.
* Use case-ul: reprezinta un set de actiuni realizate de un sistem, avand un anumit scop.
* Componenta: descrie partea fizica a unui sistem.
* Nod: reprezinta un element fizic ce exista in run time.

b. Comportamentale, ce constau in partile dinamice ale diagramelor UML.

Exemple de obiecte comportamentale:

* Interactiunea: defineste un comportament ce consta intr-un set de mesaje transmise pentru a realiza anumite task-uri.
* Mecanisme de stare: sunt utile atunci cand starea unui obiect este importanta in ciclul sau de viata. Mecanismele de sare definesc niste secvente de stari pe care un obiect le parcurge, in urma unor evenimente.

c. De grupare, ce pot fi definite ca un mecanism pentru agrupa elementele unui model UML.

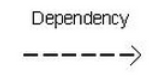
* Pachetul: este singura posibilitate de grupare a obiectelor structurale si comportamentale.

d. De adnotare, ce pot fi definite drept mecanisme de manipulare a descrierilor si comentariilor modelelor UML. Nota este singurul obiect adnotational.

* Nota: este folosita pentru adaugarea de comentarii, constrangeri, etc. a unui element UML.

2. Relatiile, alte blocuri UML fundamentale. Ele arata feul in care elementele sunt asociate intre ele, descriind in acest fel functionalitatea aplicatiei.

Exista patru tipuri de relatii:

1. Dependente: relatii intre doua obiecte, in care unul il afecteaza pe celalalt.
2. Asocieri: un set de legaturi ce conecteaza elementele unui model UML. Ele descriu felul in care obiectele participa la relatia respectiva.
3. Generalizari: relatii ce conecteaza un element special cu unul general. Practic, ele descriu mostenirile intre obiecte.
4. Realizari: relatii ce conecteaza doua elemente. Unul descrie o responsabilitate ne-implementata si celalalt o implementeaza. Aceasta relatie este folosita pentru interfete.

3. Diagramele: toate elementele, relatiile alcatuiesc o diagrama UML completa, iar diagrama reprezinta un sistem.

Tipuri de diagrame:

1. **Diagrama de clasa**

Este o diagrama statica si reprezinta imaginea statica a aplicatiei. Diagrama de clasa nu este folosita doar pentru vizualizarea, descrierea si documentarea diferitelor aspecte ale unui sistem, dar si pentru a construi cod executabil pentru aplicatiile software.

Diagrama de clasa descrie atributele si operatiile unei clase, dar si constrangerile impuse de sistem. Acestea sunt utilizate frecvent pentru modelarea sistemelor orientate pe obiect, deoarece acestea sunt singurele diagrame UML care pot fi modelate cu ajutorul unui limbaj orientat pe obiect.

Scopul unei diagrame de clasa este:

* analiza si designul imaginii statice a aplicatiei
* descrierea rolurilor unui sistem
* baza pentru diagramele de componente si de instalare

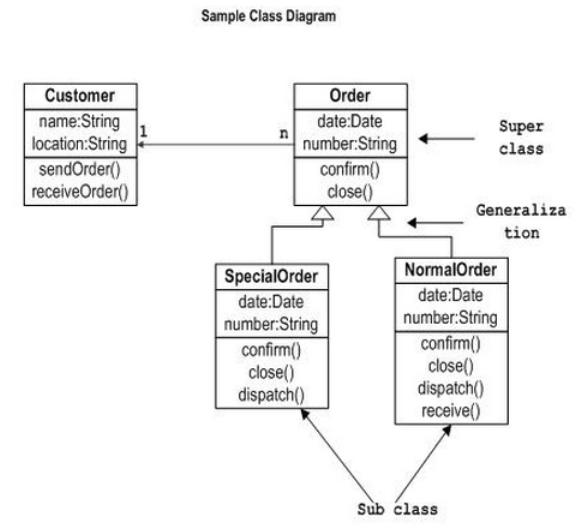
Diagramele de clasa sunt cele mai importante diagrame UML, folosite pentru realizarea aplicatiilor software, de aceea este foarte importanta desenarea corecta a acestora.

O colectie de diagrame de clasa reprezinta intregul sistem.

Reguli de desenare a diagramei:

* numele diagramei trebuie sa fie reprezentativ pentru a descrie aspectele sistemului
* fiecare element si relatiile acestuia trebuiesc pre identificate
* rolurile (atribute si metode) fiecarei clase trebuiesc pre identificate
* pentru fiecare clasa trebuie specificat minimul de proprietati, deoarece proprietatile inutile vor complica diagrama
* se folosesc note ori de cate ori este necesara descrierea vreunui aspect al diagramei, deoarece trebuie sa fie inteles de catre dezvoltator

Exemplu de diagrama de clasa:



* Order si Customer sunt identificati ca doua elemente ale sistemului si au o relatie de unu-la-mai-multi, deoarece un client poate avea mai multe comenzi
* Clasa Order este o clasa abstracta, iar clasele SpecialOrder si NormalOrder o mostenesc, avand aceleasi proprietati
* In plus, SpecialOrder si NormalOrder au metode aditionale ca dispach() si receive()

1. **Diagrama de obiect**

Este derivata din diagrama de clasa, fiind dependente de acestea.Diagrama de obiect reprezinta o instanta a diagramei de clasa.

Diagramele de obiect sunt folosite pentru a randa un set de obiecte si relatia acestora ca instante. Acestea sunt mai apropiate de comportamentul sistemului. Scopul este acela de a capta imaginea statica a sistemului intr-un anumit moment.

Scopul unei diagrame de obiect este:

* relatia obiectelor sistemului
* imaginea statica a interactiunii
* intelegerea comportamentelor unui obiect si relatiile acestora dintr-o perspectiva practica

Reguli de desenare a diagramei:

In diagramele de clasa elementele sunt intr-o forma abstracta si reprezinta o schita, iar diagramele de obiect sunt intr-o forma concreta si reprezinta obiectul din lumea reala. Pentru a capta un sistem, numarul de clase este limitat, insa numarul diagramelor de obiect este nelimitat, deci doar instantele sunt considerate ca avand un impact asupra sistemului.

* Se analizeaza sistemul si se decide ce instante au date si asocieri importante
* Se iau in considerare doar acele instante care acopera functionalitatile
* Se efecutueaza optimizari, avand in vedere faptul ca numarul de instante este nelimitat

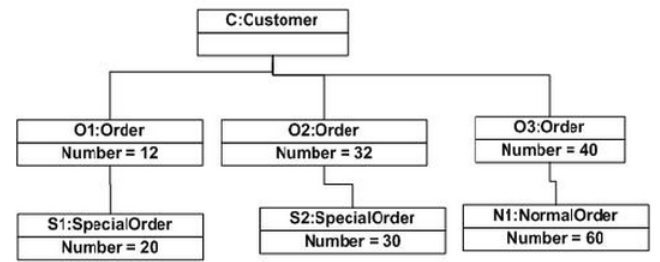
Inainte de a desena diagramele de obiect se iau in considerare urmatoarele:

* Diagramele de obiecte constau in obiecte
* Legatura in diagrame conecteaza obiectele
* Obiectele si legaturile sunt doua elemente folosite pentru a construi diagramele de obiecte

Atunci cand se deseneaza o diagrama de obiect, este importat:

* sa existe un nume reprezentativ ce indica scopul acestuia
* trebuiesc identificate cele mai importante elemente
* trebuie clarificata asocierea dintre obiecte
* valoarea diferitelor elemente trebuie captata pentru a fi inclusa in diagrama
* adaugarea notelor unde este necesara clarificarea anumitor aspecte

Exemplu de diagrama de obiect:



* Reprezinta sistemul de management al unei comenzi, prezentata in diagrama de clasa
* Este o instanta a sistemului intr-un anumit moment al achizitionarii.
* Contine urmatoarele obiecte: Customer, Order, SpecialOrder, NormalOrder

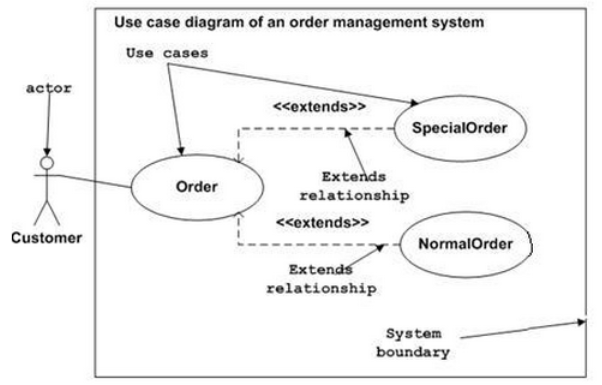
1. **Diagrame Use Case**

In diagramele UML sunt patru diagrame disponibile pentru a reprezenta modelul dinamic in natura, use case fiind una dintre acestea.

Scopul diagramei Use Case este de a capta aspectul dinamic al sistemului, pentru a aduna cerintele unui sistem, incluzand influentele interne si externe.

Diagramele Use Case reprezinta functionalitatile sistemului si de asemenea, definesc actorii care interactioneaza cu sistemul.

Exemplu:

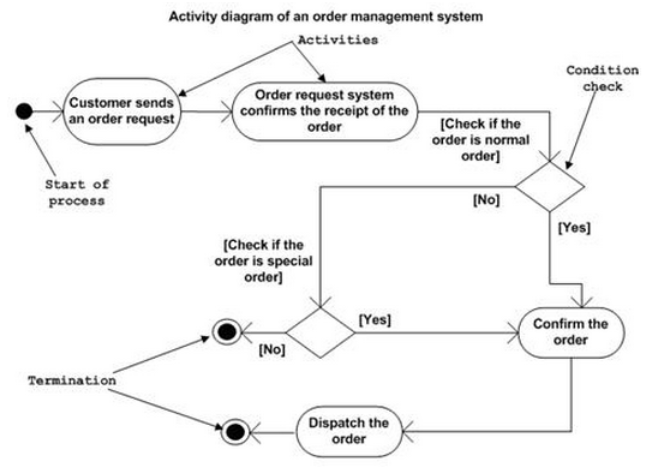


* Este un use case simplu ce reprezinta managementul sistemului unei comenzi
* Sunt reprezentate trei cazuri(Order, SpecialOrder si NormalOrder) si un actor ce reprezinta cumparatorul

1. **Diagrame de secventa**
2. **Diagrame de colaborare**
3. **Diagrame de activitate**

Descrie aspectele dinamice ale sistemului, fiind o diagrama de fluxuri ce reprezinta fluxul de la o activitate la alta.

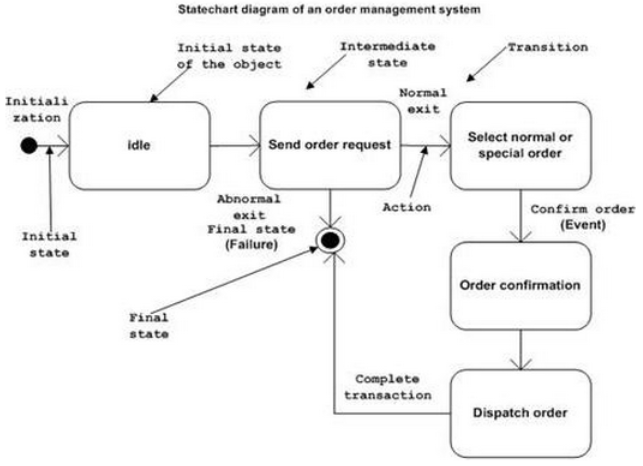
Exemplu:



1. **Diagrame Statechart**

Reprezinta diferitele stari specifice unei componente/obiect dintr-un sistem.

Exemplu:



1. **Diagrame de dezvoltare**
2. **Diagrama de componente**

Este folosita pentru a modela aspectele fizice ale sistemului. Aspectele fizice sunt elelmente ca executabile, librarii, fisiere, documente care se regasesc intr-un nod.

Diagramele de componente sunt folosite pentru a vizualiza organizarea si relatiile dintre componentele unui sistem.

Acestea nu descriu functionalitatile unui sistem, insa descriu componentele folosite pentru a realiza aceste functionalitati.

Scopul diagramelor de componente:

* vizualizarea componentelor sistemului
* construirea de executabile
* descrierea organizatiei si relatiile dintre componente

Reguli de realizare a diagramelor:

Initial, sistemul este descris folosind diferite diagrame UML, dupa care, cand artefactele sunt pregatite, diagrama de componente este folosita pentru a se obtine o idee despre implementare.

* Se foloseste in nume reprezentativ pentru a identifica acele componente
* Pregatirea unui layout imaginar inainde de a-l realiza folosing uneltele
* Folosirea notelor pentru clarificarea punctelor importante

Exemplu:

