**Tema**

**• Instalarea mediului de lucru Visual Paradigm Community Edition**.

**• Ce este UML și pentru ce este utilizat?**

Astăzi, UML este unul dintre cele mai răspândite limbaje grafice de modelare orientate pe obiect. În ciuda numeroaselor revizuiri, rădăcinile sale (metoda Booch, OMT, OOSE) sunt încă clar recunoscute. UML este potrivit pentru modelarea relațiilor complexe între obiecte și a proceselor cu concurență. UML este un limbaj de modelare de uz general, ceea ce înseamnă că utilizarea sa nu este restricționată la o anumită zonă de aplicație. Furnizează concepte de limbaj și modelare, precum și o notare grafică intuitivă pentru modelarea diverselor domenii de aplicație, permițând specificarea, proiectarea, vizualizarea și documentarea unui sistem software. Rezultatul modelării cu UML este un model grafic care oferă diferite vederi ale unui sistem sub formă de diverse diagrame.

UML nu este legat de un instrument specific de dezvoltare, de un limbaj de programare specific sau de o platformă țintă specifică pe care sistemul trebuie să fie utilizat. UML nu oferă nici un proces de dezvoltare software. De fapt, UML separă limbajul de modelare și metoda de modelare. Aceasta din urmă poate fi definită în funcție de proiect sau de companie. Cu toate acestea, conceptele de limbaj ale UML favorizează un proces iterativ și incremental.

Utilizarea în dezvoltarea software UML poate fi utilizat în mod consecvent pe întregul proces de dezvoltare software. La toate etapele dezvoltării, aceleași concepte de limbaj pot fi utilizate în aceeași notație. Astfel, un model poate fi rafinat în etape. Nu este necesar ca un model să fie tradus într-un alt limbaj de modelare. Acest lucru permite un proces de dezvoltare software iterativ și incremental. UML este potrivit pentru diverse domenii de aplicație cu cerințe diferite referitoare la complexitate, volum de date, timp real, etc.

Elementele modelului UML și utilizarea corectă a acestora sunt specificate în metamodelul UML generic. Conceptele de limbaj sunt definite într-un mod atât de generic încât se obține o aplicabilitate largă și flexibilă. Pentru a evita restrângerea utilizării UML, standardul este (intenționat) vag în diverse puncte, permițând interpretări diferite sub formă de puncte de variație semantică. Cu toate acestea, acest lucru are și o latură mai puțin favorabilă; conduce și la diferite implementări ale standardului de limbaj de către instrumente de modelare, ceea ce face, din nefericire, dificilă schimbul de modele.

**• Ce sunt modelele și care este utilitatea lor?**

Modelele ne permit să descriem sistemele eficient și elegant. În 1973, Herbert Stachowiak a propus o teorie a modelelor care se distinge prin trei caracteristici:

1. Mapare: un model este întotdeauna o imagine (mapare) a ceva, o reprezentare a originalelor naturale sau artificiale care pot fi în sine modele.

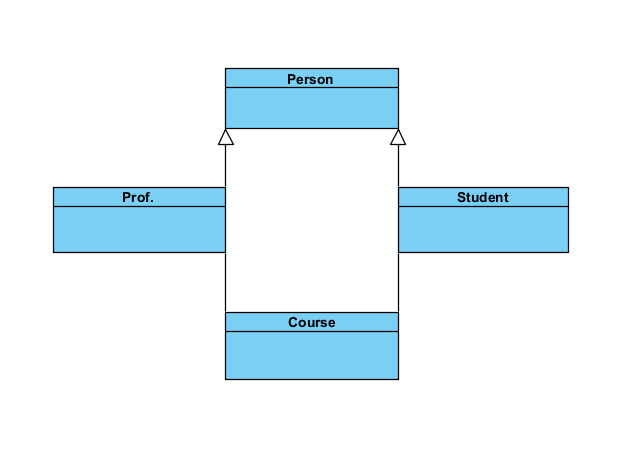
2. Reducere: un model nu capturează toate atributele originalelor, ci doar pe cele care par relevante pentru modelator sau utilizatorul modelului.

3. Pragmatism: pragmatismul înseamnă orientare către utilitate. Un model este atribuit unui original pe baza următoarelor întrebări: Pentru cine? De ce? Pentru ce? Un model este utilizat de către modelator sau utilizator în locul originalului într-un interval de timp specific și pentru un scop specific.

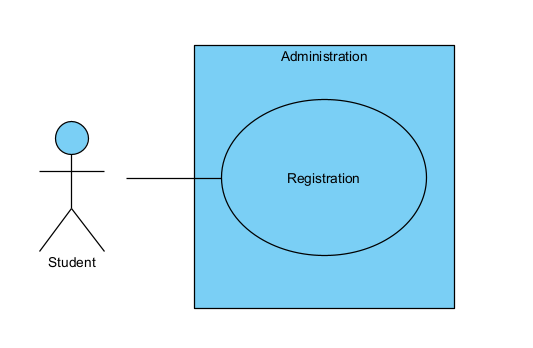
Modelele sprijină o reprezentare a unui sistem redus la esențiale pentru a minimiza complexitatea sistemului la aspecte gestionabile. De obicei, un sistem nu este descris printr-o singură vedere, ci printr-un număr de vederi care împreună produc o imagine generală unificată. Astfel, o vedere ar putea descrie obiectele implicate și relațiile dintre ele; o altă vedere ar putea descrie comportamentul unui grup de obiecte sau prezenta interacțiunile dintre diferite obiecte.

**• Familiarizarea cu mediul de lucru mai sus amintit prin crearea anumitor diagrame din sectiunile 2.3.1 și 2.3.2 din [1]. Mai exact:**

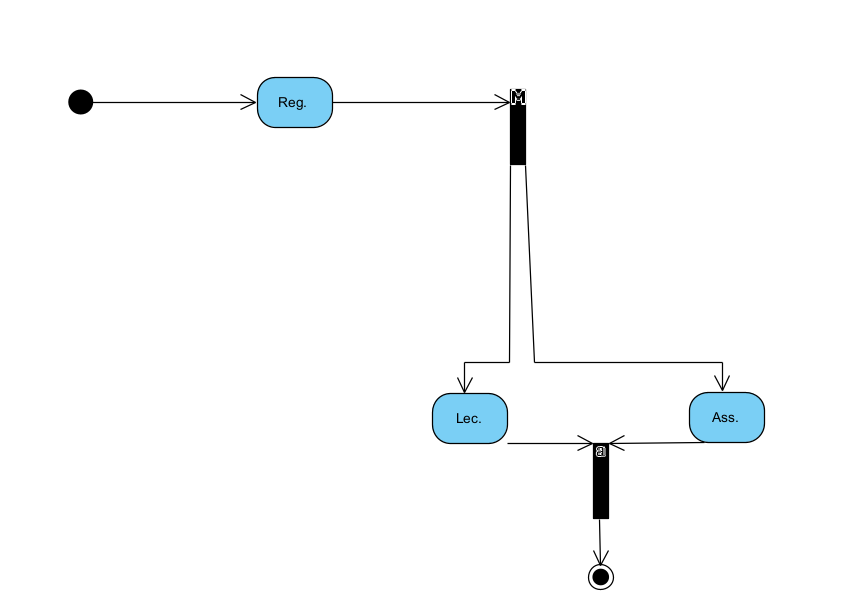
**– Class diagram – diagrama de clasă**



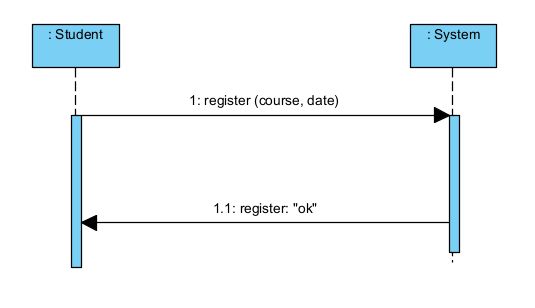
**– Use case diagram – diagrama de cazuri de utilizare**

****

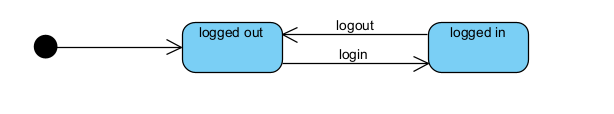
**– Activity diagram – diagrama de activitate**

****

**– Sequence diagram – diagrama de secvente**

****

**– State machine diagram – diagrama mașinii de stare**

****

**• Ce tipuri de diagrame sunt prezentate în [1] – Capitolul 2? Care sunt asemănarile și deosebirile între acestea?**

Diagrama de caz de utilizare (Use Case Diagram): Specifică funcționalitatea de bază a unui sistem software.

Diagrama de clasă (Class Diagram): Defineste ce obiecte sau ce clase sunt implicate în realizarea acestei funcționalități.

Diagrama de stări (State Machine Diagram): Defineste comportamentul intra-obiect.

Diagrama de secvență (Sequence Diagram): Specifică comportamentul inter-obiect.

Diagrama de activitate (Activity Diagram): Permite definirea proceselor care "implementează" cazurile de utilizare din diagrama de caz de utilizare.

**Diagrama de Caz de Utilizare vs. Diagrama de Clasă:**

Asemănări:

* Ambele sunt diagrame statice care oferă o vedere la nivel înalt a sistemului.
* Sunt utilizate pentru a comunica și înțelege structura și funcționalitățile sistemului.

Deosebiri:

* Diagrama de Caz de Utilizare: Se concentrează pe interacțiunile dintre sistem și actorii săi externi, evidențiind funcționalitățile sistemului.
* Diagrama de Clasă: Se concentrează pe structura internă a sistemului, evidențiind clasele, atributele și relațiile dintre ele.

**Diagrama de Stare vs. Diagrama de Secvență:**

Asemănări:

* Ambele sunt diagrame de comportament și descriu modul în care sistemul reacționează la evenimente sau la interacțiuni.

Deosebiri:

* Diagrama de Stare: Se concentrează pe comportamentul intra-obiect, evidențiind stările posibile ale unui obiect și tranzițiile între aceste stări.
* Diagrama de Secvență: Se concentrează pe comportamentul inter-obiect, ilustrând ordinea cronologică a interacțiunilor dintre obiecte.

**Diagrama de Activitate vs. Diagrama de Caz de Utilizare:**

Asemănări:

* Ambele sunt utilizate pentru a modela procesele și fluxurile de lucru în cadrul sistemului.

Deosebiri:

* Diagrama de Activitate: Se concentrează pe fluxurile de control și procesele care implementează cazurile de utilizare.
* Diagrama de Caz de Utilizare: Se concentrează pe interacțiunile între sistem și actorii săi, evidențiind funcționalitățile sistemului.

**• Caracterizați succint fiecare diagrama prezentată în [1] – Capitolul 2 (cele 5 menționate mai sus)**

**Diagrama de Caz de Utilizare (Use Case Diagram):**

Scop:

* Reprezintă interacțiunile dintre sistem și actorii săi (utilizatori sau alte entități externe).
* Explorează cerințele și funcționalitățile sistemului într-o manieră intuitivă.

Caracteristici:

* Utilizează actori pentru a reprezenta roluri sau entități externe și cazuri de utilizare pentru a descrie funcționalități.
* Indică relațiile dintre actori și cazurile de utilizare.
* Este o unealtă eficientă pentru a comunica cerințele de bază ale sistemului într-un mod ușor de înțeles.

**Diagrama de Clasă (Class Diagram):**

Scop:

* Ofereă o perspectivă asupra structurii statice a sistemului, evidențiind clasele, atributele și relațiile dintre ele.
* Abordează conceptele de obiecte și modul în care acestea interacționează.

Caracteristici:

* Folosește clase pentru a reprezenta tipurile de obiecte și asociațiile dintre ele.
* Include atribute pentru a descrie caracteristicile claselor și metode pentru a indica comportamentul lor.
* Permite vizualizarea detaliilor structurale ale sistemului, precum și a relațiilor dintre diverse entități.

**Diagrama de Stare (State Machine Diagram):**

Scop:

* Explorează stările posibile și tranzițiile între acestea pentru un obiect în timpul vieții sale.
* Util pentru modelarea comportamentului obiectelor în funcție de evenimente.

Caracteristici:

* Folosește stări, tranziții, evenimente și acțiuni pentru a ilustra comportamentul unui obiect în timp.
* Permite vizualizarea ciclului de viață al obiectelor și modul în care acestea răspund la evenimente.

**Diagrama de Secvență (Sequence Diagram):**

Scop:

* Arată interacțiunile cronologice dintre obiecte într-un scenariu specific.
* Explorează ordinea exactă a mesajelor trimise între obiecte.

Caracteristici:

* Utilizează obiecte, linii de viață și mesaje pentru a ilustra modul în care obiectele cooperează.
* Indică detalii despre colaborarea și secvența de acțiuni între obiecte în timp real.

**Diagrama de Activitate (Activity Diagram):**

Scop:

* Reprezintă procese sau fluxuri de lucru, inclusiv decizii și bifurcări.
* Explorează detaliile unui proces sau activitate.

Caracteristici:

* Utilizează noduri pentru a reprezenta acțiuni, decizii, bifurcări și alte elemente de control al fluxului.
* Indică ordinea logică a acțiunilor și deciziilor în cadrul unei activități.
* Este eficient pentru a descrie procese de afaceri, algoritmi sau fluxuri de lucru în detaliu.

**Diagrama de Obiect (Object Diagram):**

Scop:

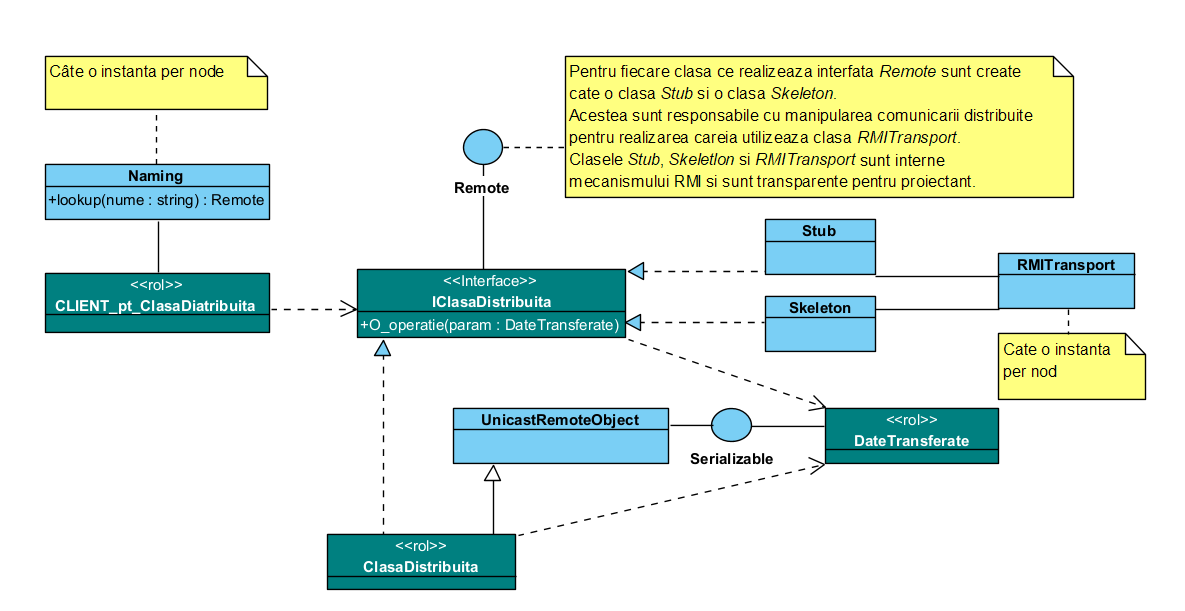
* Furnizează o vedere statică asupra unui sistem, prezentând instanțele concrete ale claselor și relațiile dintre acestea la un anumit moment în timp.
* Ilustrează structura detaliată a sistemului, bazată pe informații concrete.

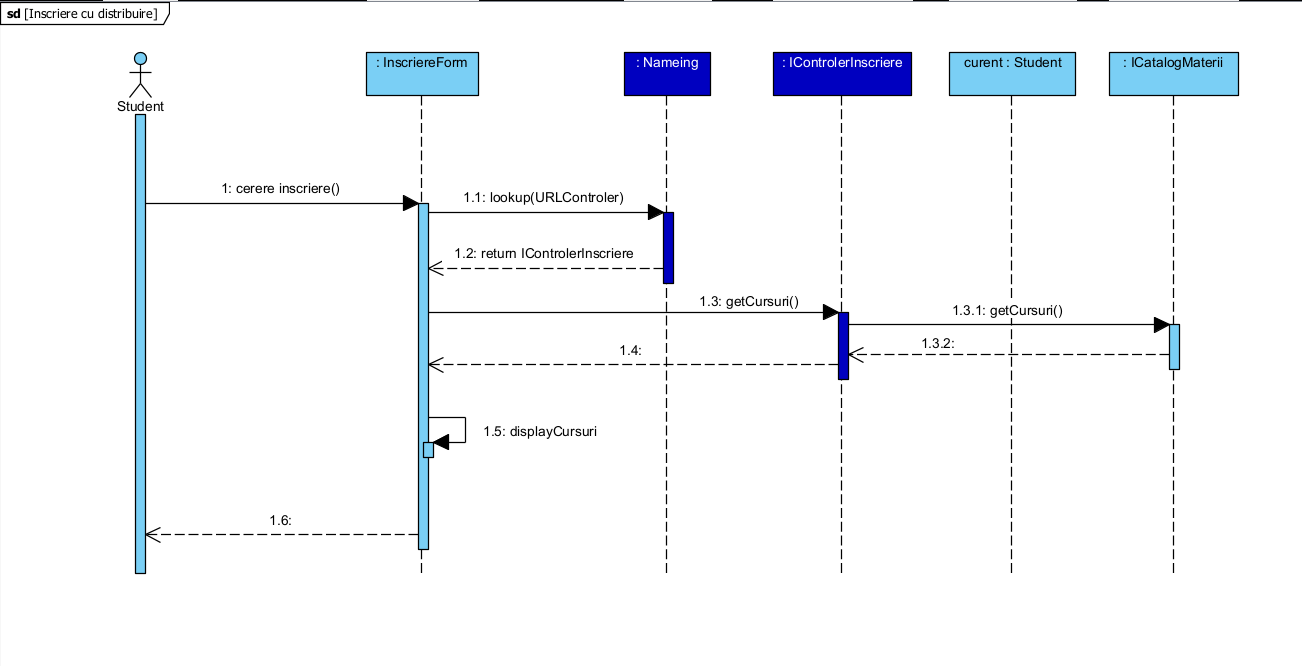
Caracteristici:

* Utilizează instanțe concrete ale claselor (obiecte) pentru a prezenta starea sistemului într-un anumit moment.
* Arată relațiile între obiecte, evidențiind modul în care acestea interacționează într-un scenariu specific.
* Oferea o imagine de ansamblu asupra instanțelor și relațiilor concrete dintre obiecte într-un context dat.

**• Folosind Visual Paradigm, realizați diagramele de clasă din Figurile 1 și 2.**

**Figure 1: O diagram de clase**

****

**Figure 2: O diagramă de secvențe**