

Ingineria Programării

Cursul 3 – 2–3 Martie
adiftene@infoiasi.ro

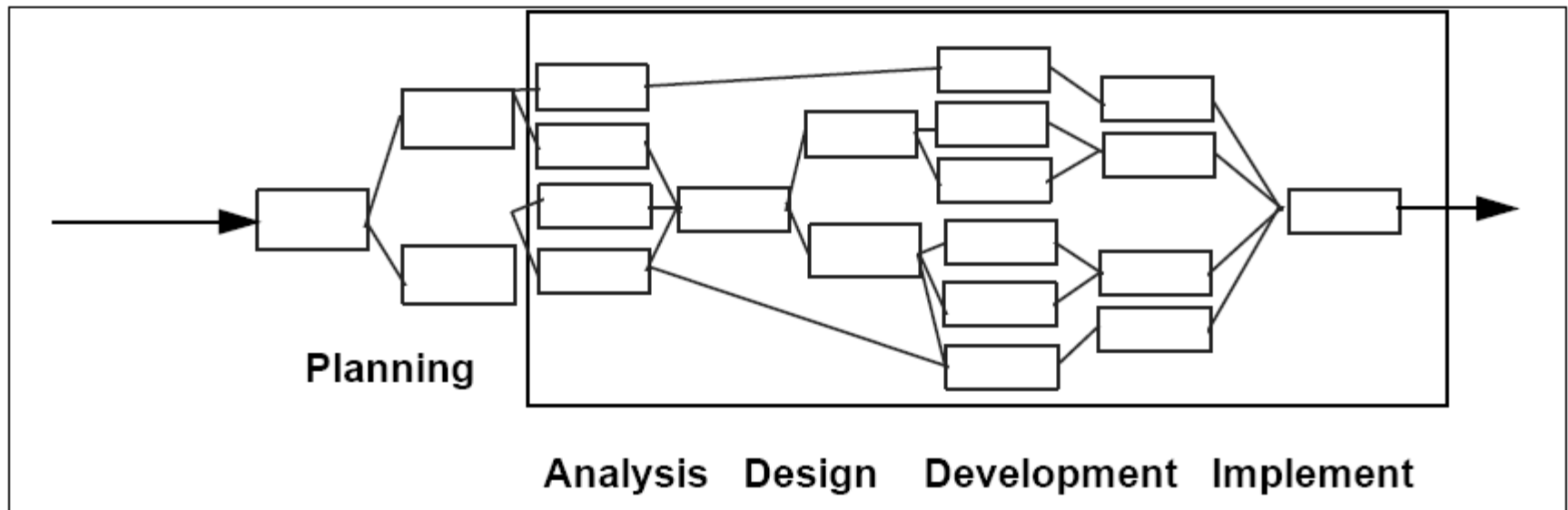
Cuprins

- ▶ Din Cursurile 1, 2...
- ▶ Modelarea
- ▶ Limbaje de Modelare
 - Limbaje Grafice
- ▶ UML – Istoric
- ▶ UML – Definiție
- ▶ UML – Tipuri de Diagrame
- ▶ UML – Diagrame Use Case
 - Actori
 - Use Case
- ▶ UML – Diagrame de Clase

Din Cursurile 1,2

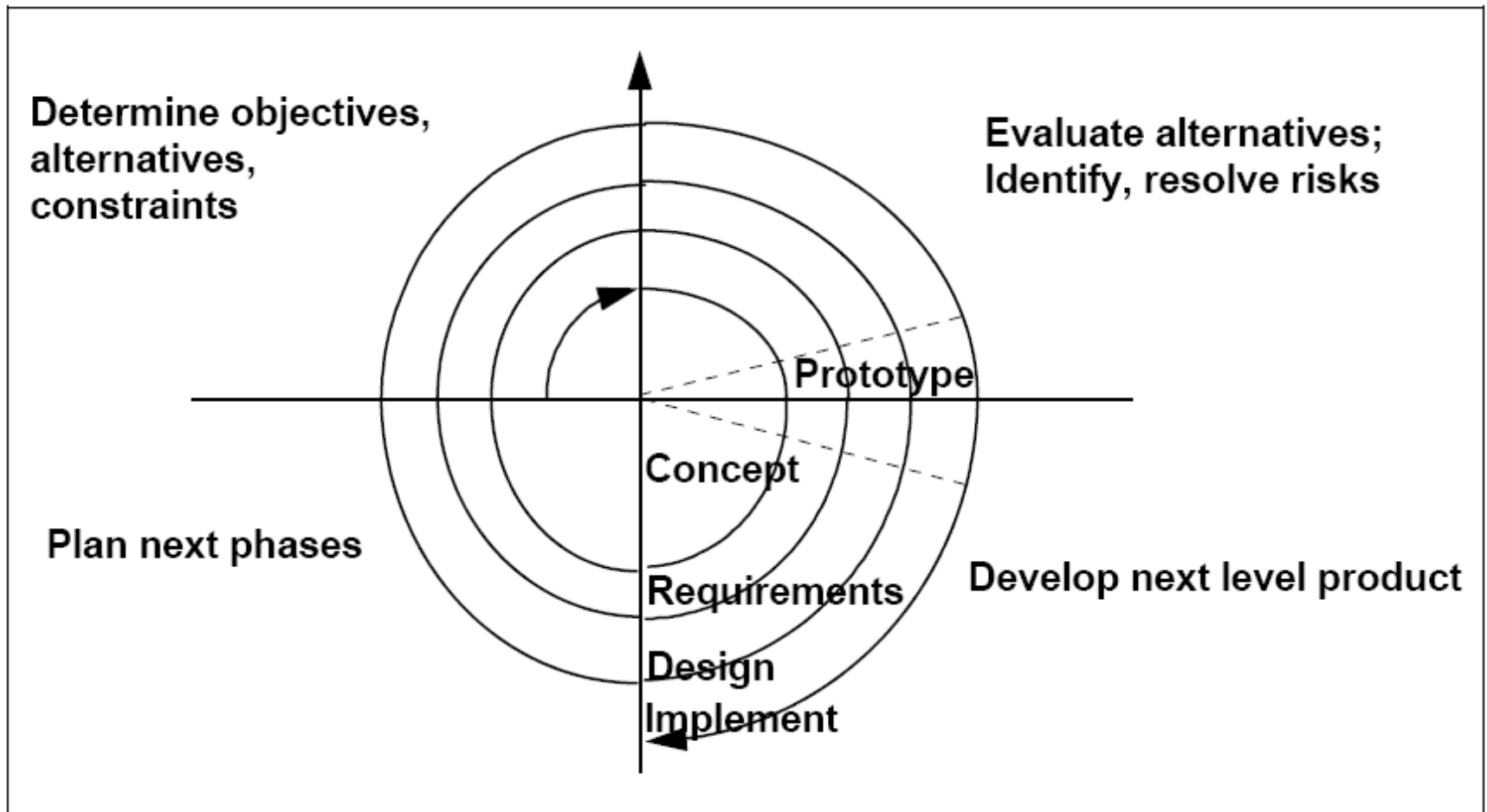
- ▶ Ingineria programării (Software engineering)
- ▶ Etapele dezvoltării programelor
- ▶ Modele de dezvoltare
- ▶ Ingineria cerințelor

Din cursul 1



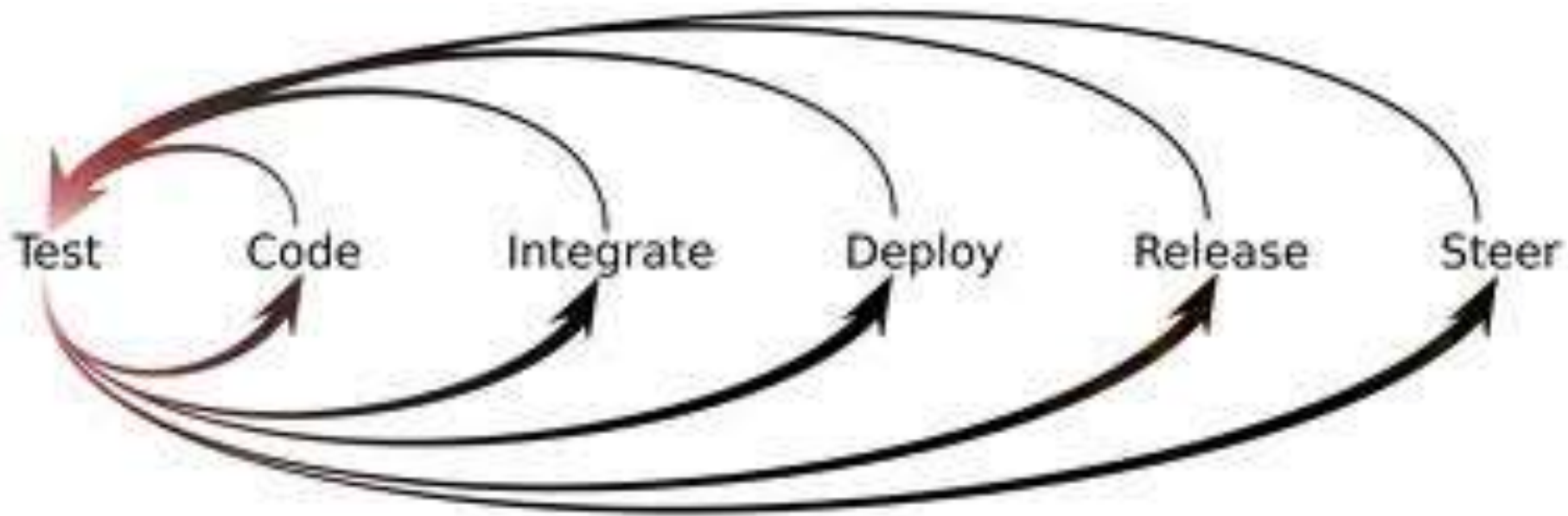
Modelul în cascadă...

Din cursul 1



Modelul în spirală

Din cursul 2



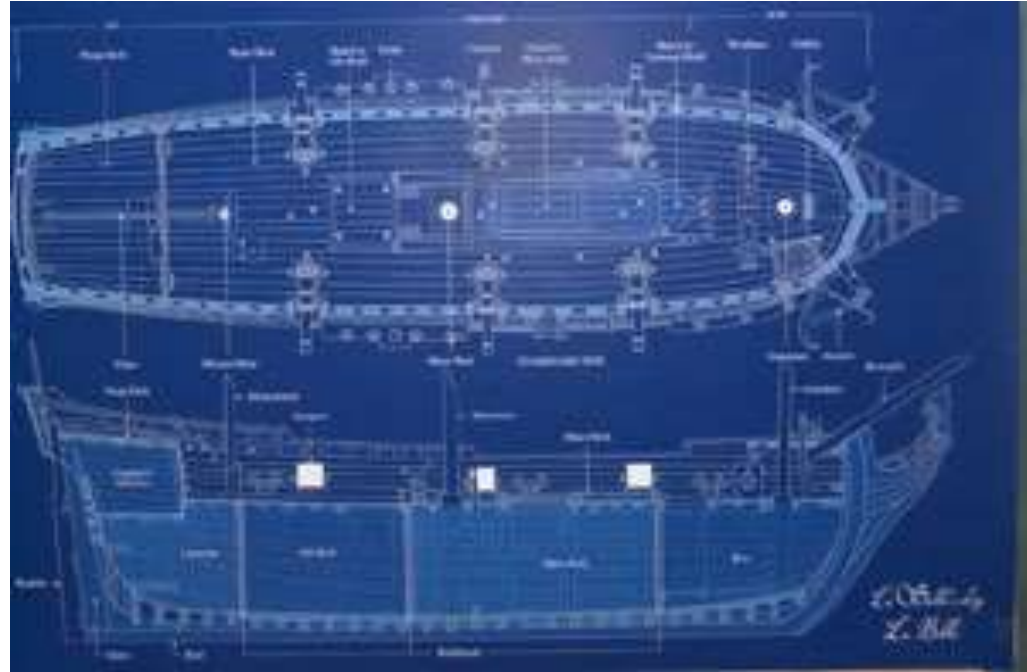
- ▶ XP, TDD...

Din cursul 2

- ▶ Ingineria cerințelor:
 - Actori
 - Scenarii
- ▶ Studiu de caz
 - Identificare actori
 - Identificare scenarii (Obiectiv, Pași, Extensii)
 - Construire diagrame de clasă

Modelarea – De ce?

- ▶ Ce este un model?
 - Simplificarea realității
 - Planul detaliat al unui sistem (blueprints)



- ▶ De ce modelăm?
 - Pentru a înțelege mai bine ce avem de făcut
 - Pentru a ne concentra pe un aspect la un moment dat
- ▶ Unde folosim modelarea?

Scopurile Modelării

- ▶ Vizualizarea unui sistem
- ▶ Specificarea structurii sale și/sau a comportării
- ▶ Oferirea unui șablon care să ajute la construcție
- ▶ Documentarea deciziilor luate

Modelarea Arhitecturii

- ▶ Cu ajutorul **Use case**-urilor: pentru a prezenta cerințele
- ▶ Cu ajutorul **Design**-ului: surprindem vocabularul și domeniul problemei
- ▶ Cu ajutorul **Proceselor**: surprindem procesele și thread-urile
- ▶ Cu ajutorul **Implementării**: avem modelarea aplicației
- ▶ Cu ajutorul **Deployment**: surprindem sistemul din punct de vedere ingineresc

Principiile Modelării

- ▶ Modelele influențează soluția finală
- ▶ Se pot folosi diferite niveluri de precizie
- ▶ Modelele bune au corespondent în realitate
- ▶ Nu e suficient un singur model

Limbaje de Modelare

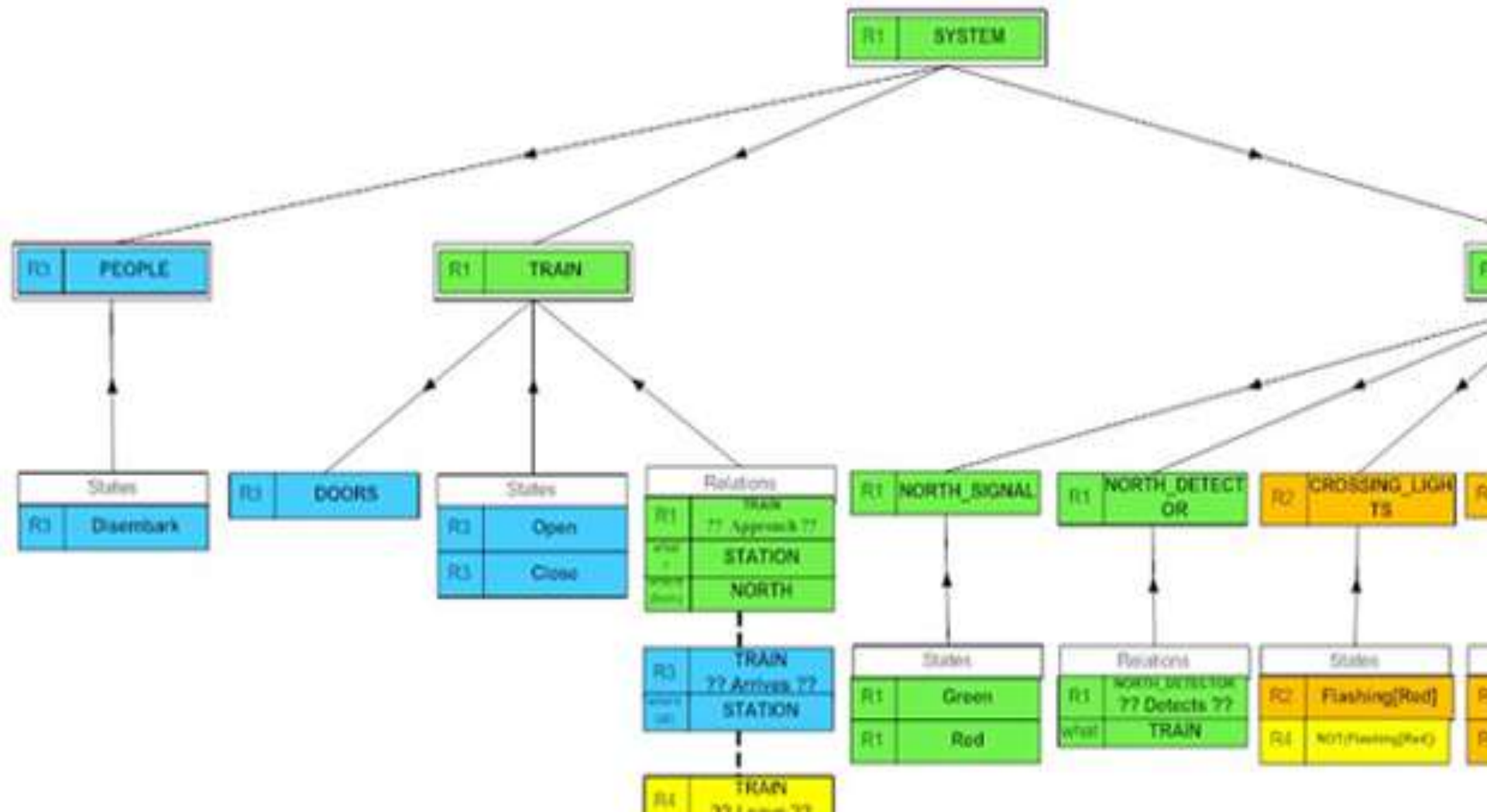
- ▶ Analiza și proiectarea unui proiect trebuie făcute înainte de realizarea codului
- ▶ În prezent, se acordă o atenție deosebită acestei etape, deoarece de ele depind **producerea și refolosirea** de software
- ▶ Pentru analiza și proiectarea programelor s-au creat **limbajele de modelare**
- ▶ **Limbaaj de modelare este** un limbaj artificial care poate fi folosit să exprime informații sau cunoaștere sau sisteme

Tipuri de Limbaje de Modelare

- ▶ **Limbaje Grafice:** arbori comportamentali, modelarea proceselor de business, EXPRESS (modelarea datelor), flowchart, ORM (modelarea rolurilor), rețele Petri, **diagrame UML**
- ▶ **Limbaje Specifice:** modelare algebrică (AML) (pentru descrierea și rezolvarea problemelor de matematică ce necesită putere computațională mare), modelarea domeniilor specifice (DSL), modelarea arhitecturilor specifice (FSML), modelarea obiectelor (object modeling language), modelarea realității virtuale (VRML)

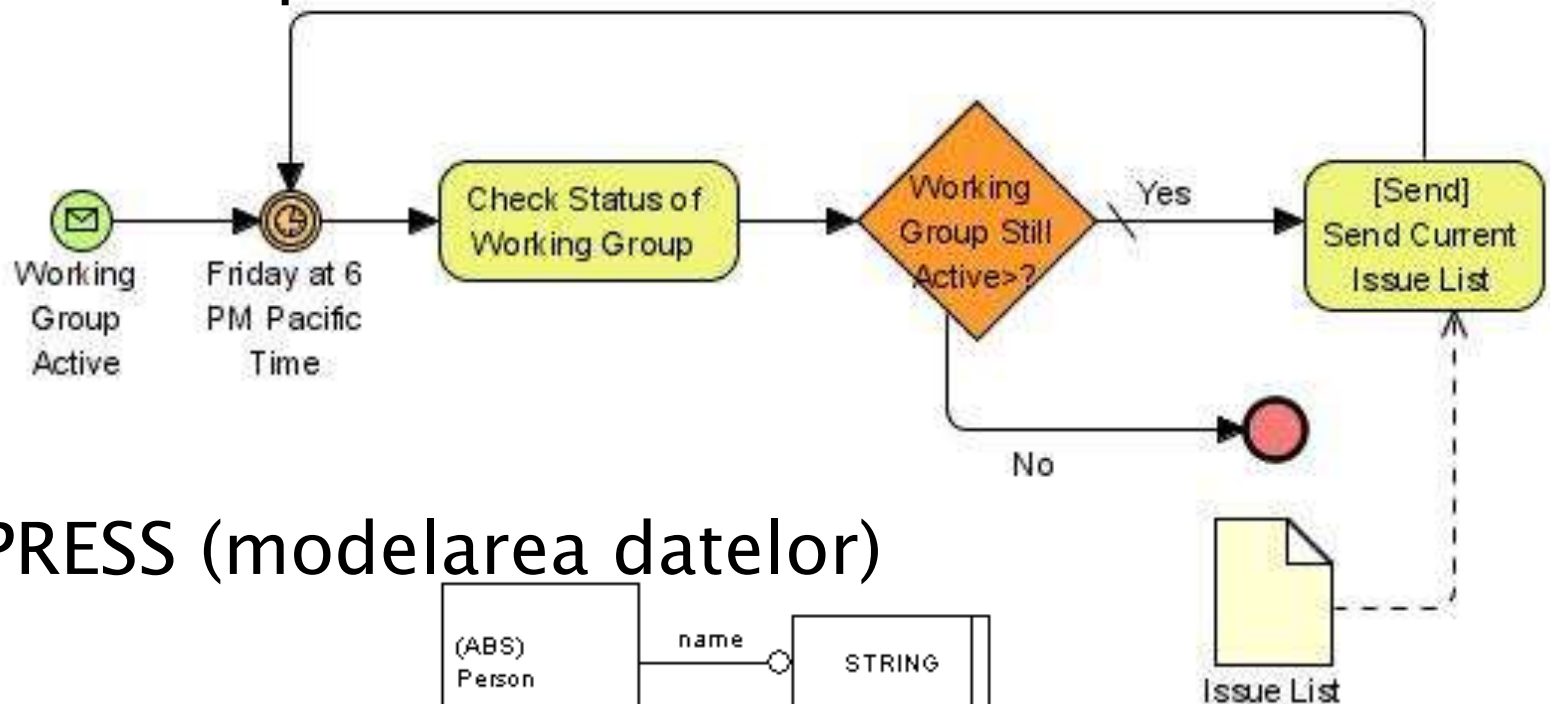
Limbaje Grafice 1

► Arbori comportamentali

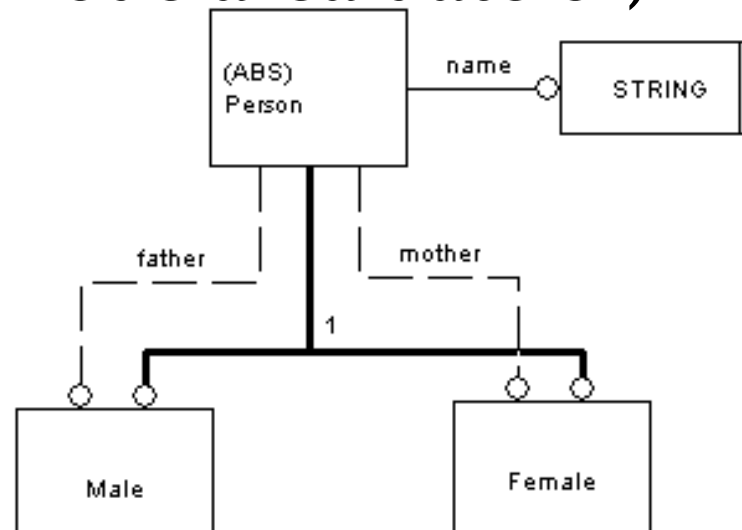


Limbaje Grafice 2

► Modelarea proceselor de business

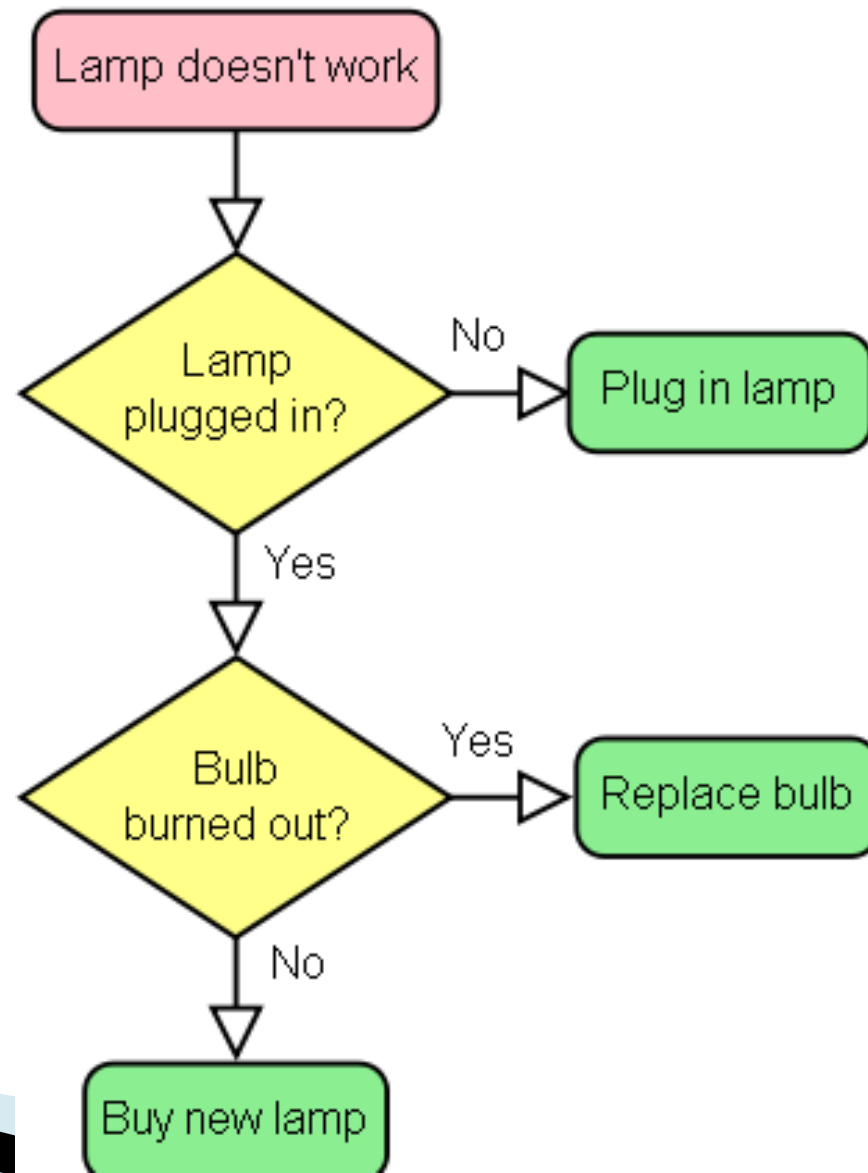


► EXPRESS (modelarea datelor)



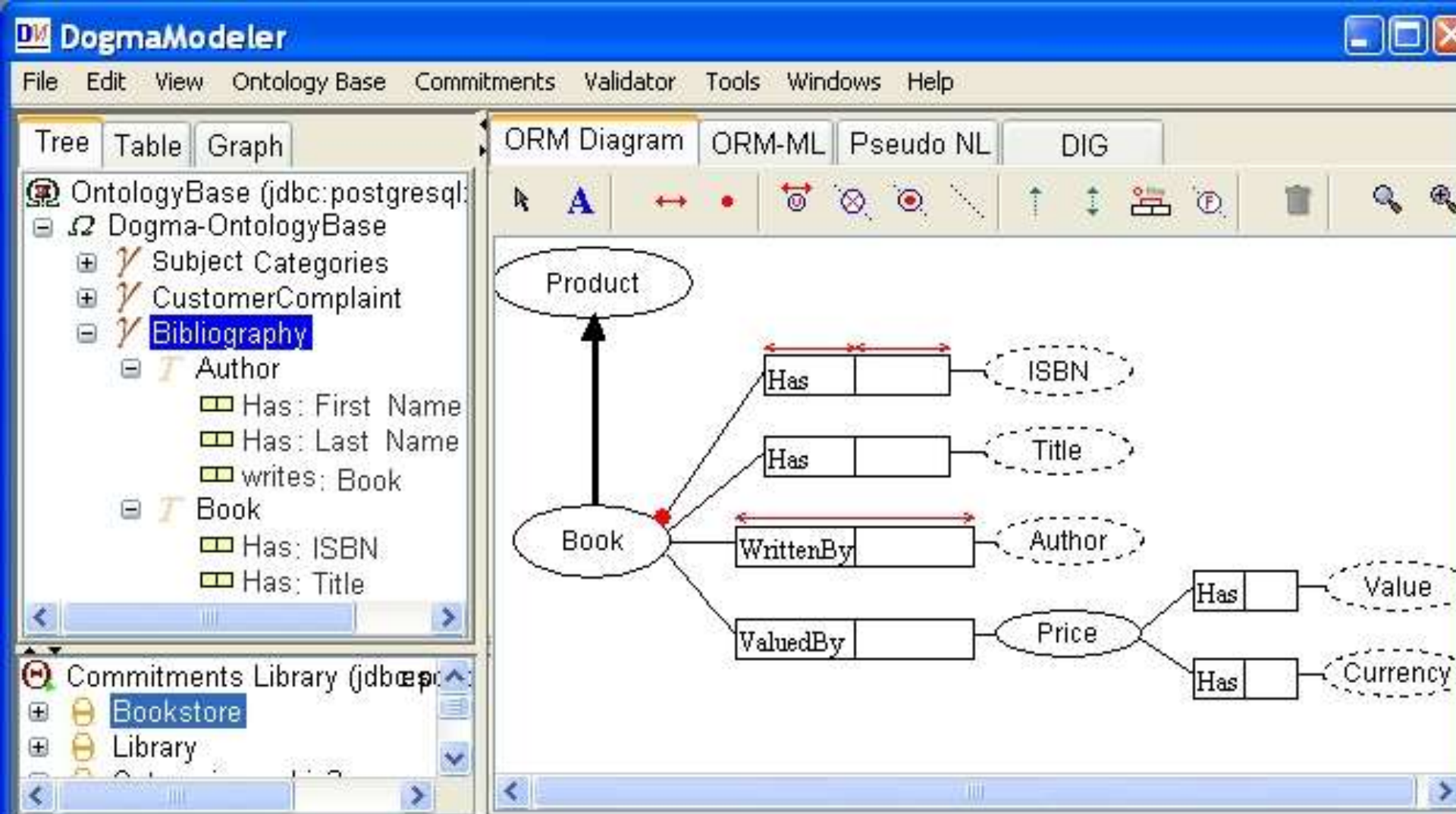
Limbaje Grafice 3

► Flowchart



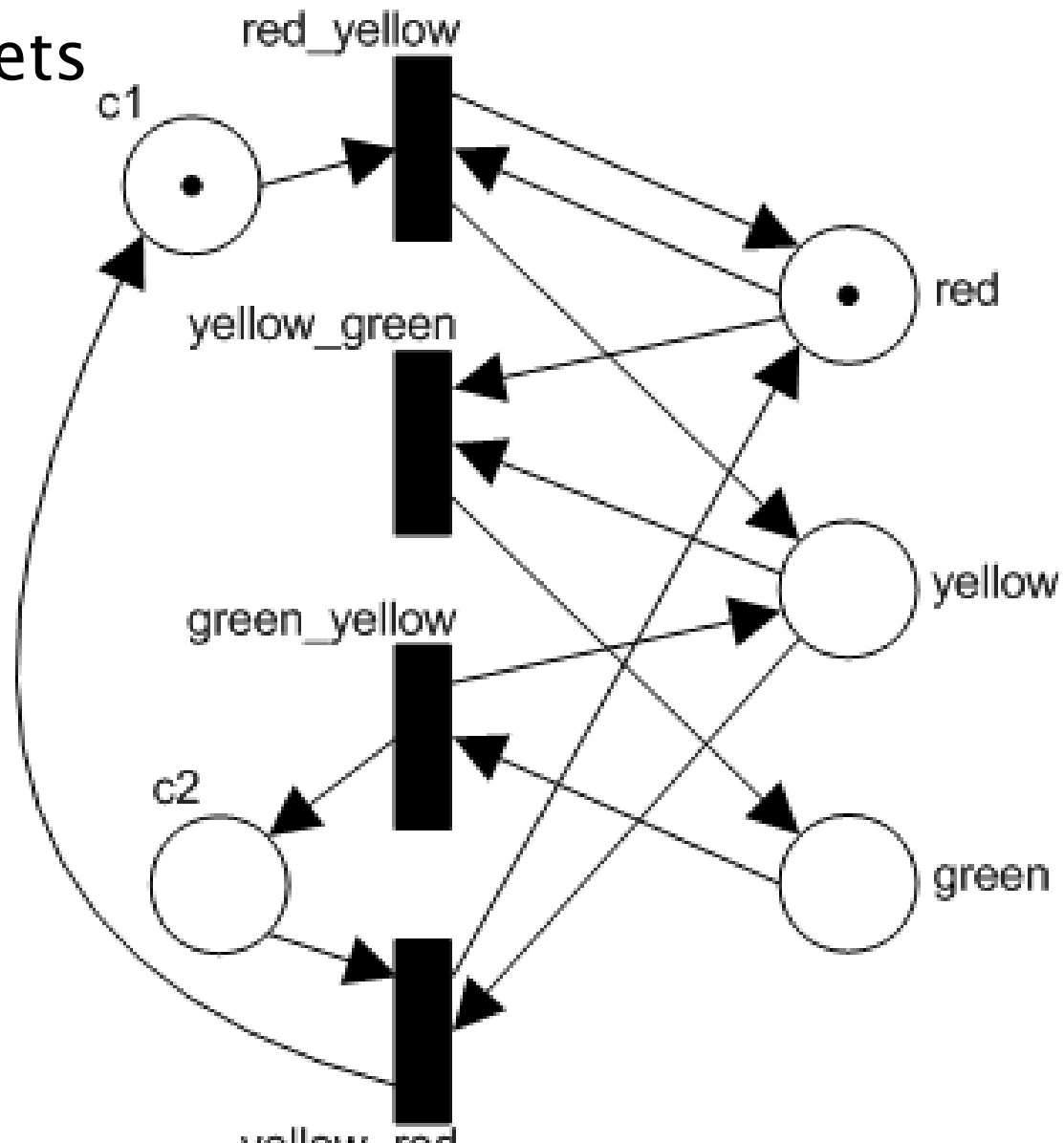
Limbaje Grafice 4

► ORM (Object Role Modeling)



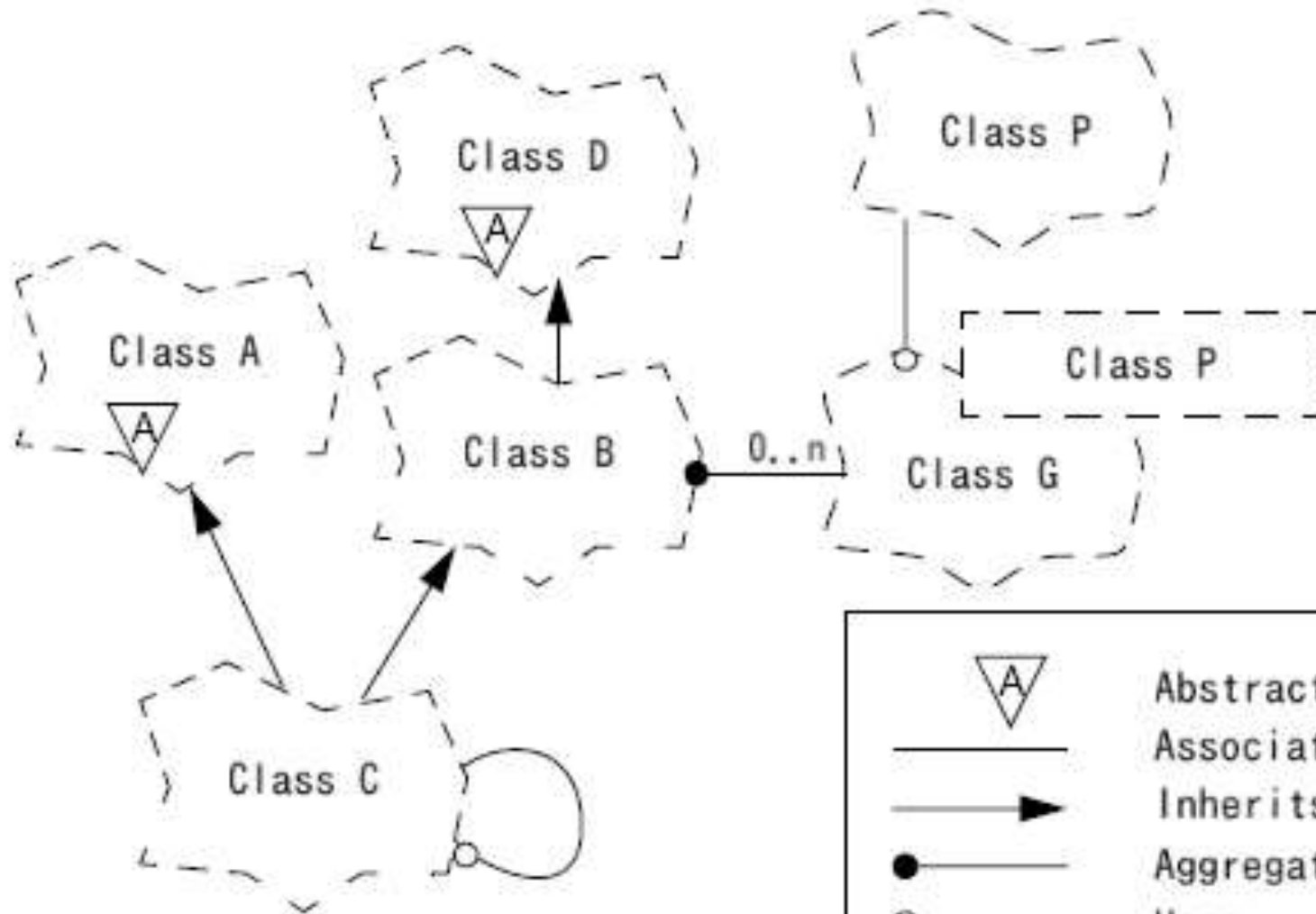
Limbaje Grafice 5

► Petri Nets



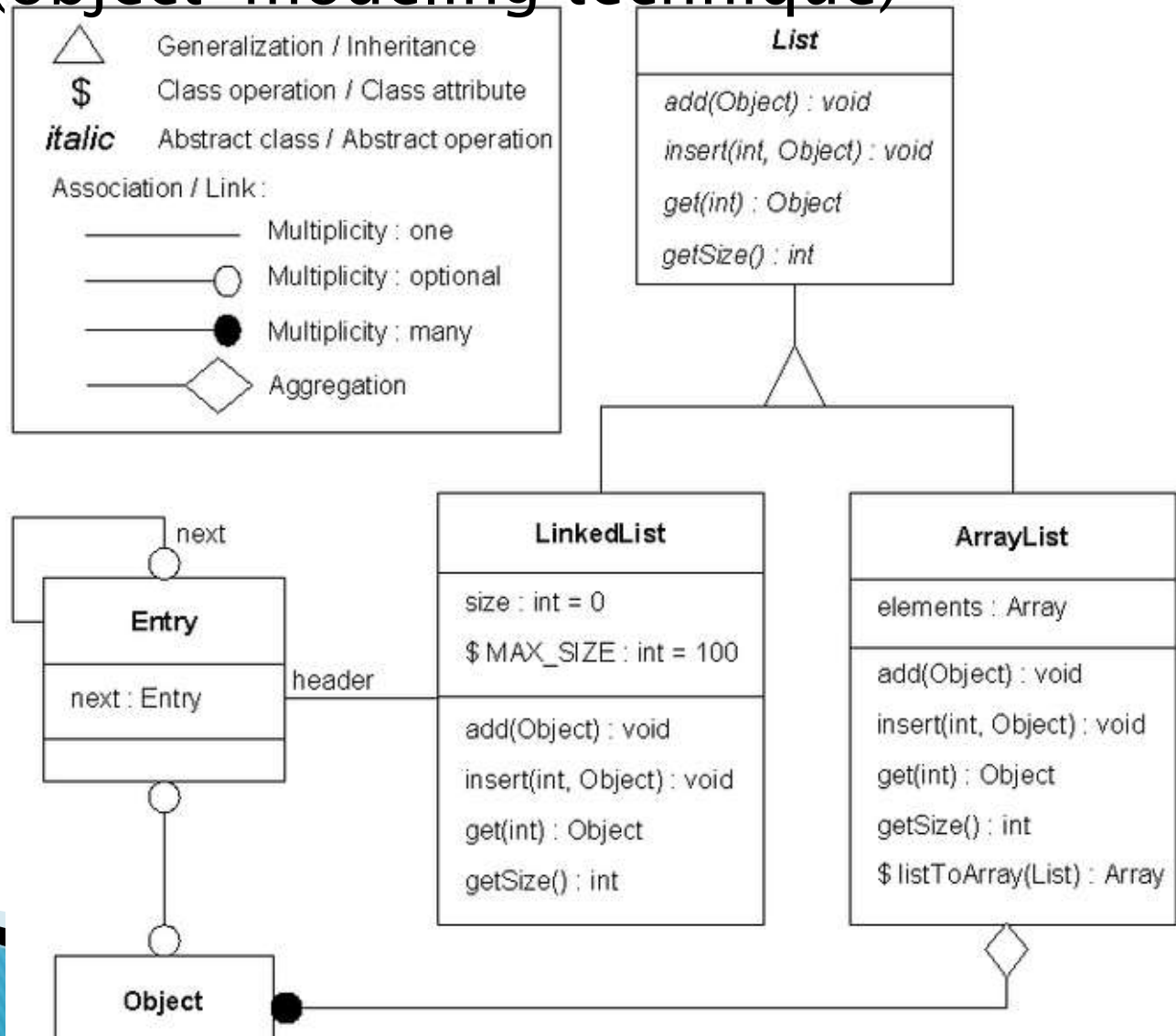
Limbaje Grafice 6

- ▶ Metoda Booch (Grady Booch) – analiza și design oo



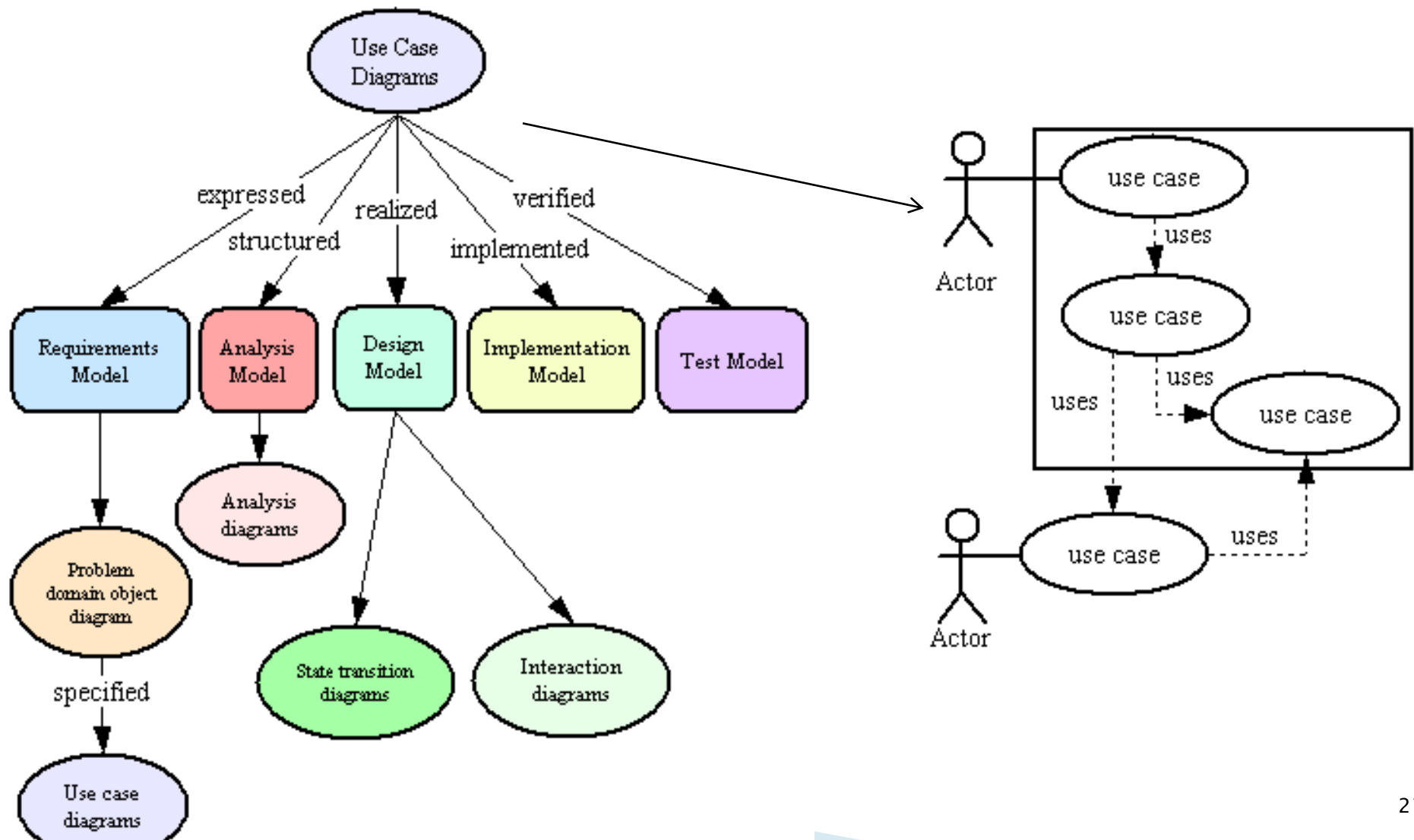
Limbaje Grafice 7

► OMT (object-modeling technique)



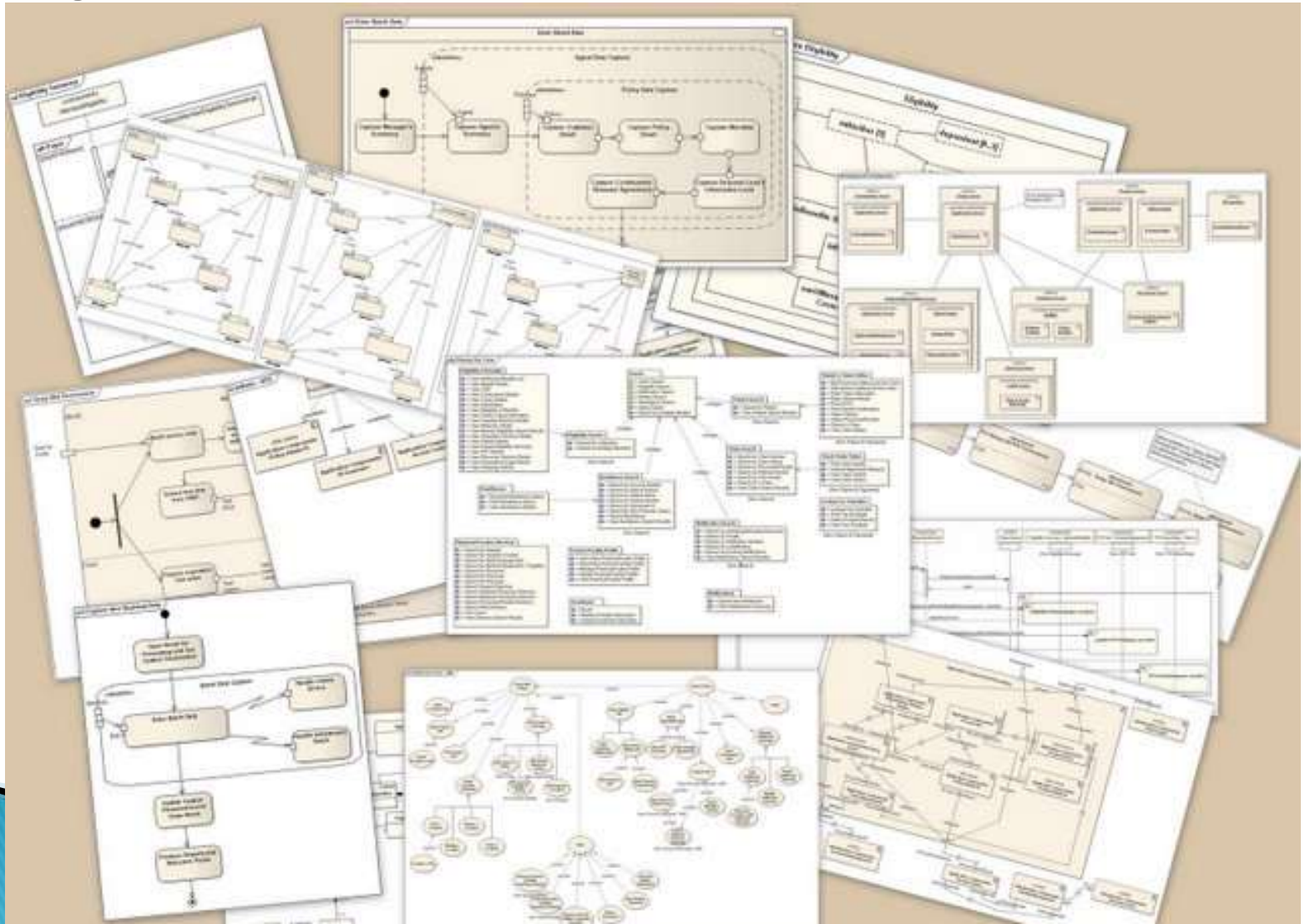
Limbaje Grafice 8

- ▶ OOSE (Object-oriented software engineering)



Limbaje Grafice 9

- Diagrame UML



UML – Introducere

- ▶ UML (Unified Modeling Language) este succesorul celor mai bune trei limbaje OO de modelare anterioare:
 - Booch (Grady Booch)
 - OMT (Ivar Jacobson)
 - OOSE (James Rumbaugh)
- ▶ UML se constituie din unirea acestor limbaje de modelare și în plus are o expresivitate mai mare

The diagram illustrates the lineage of UML and other modeling languages. The timeline is as follows:

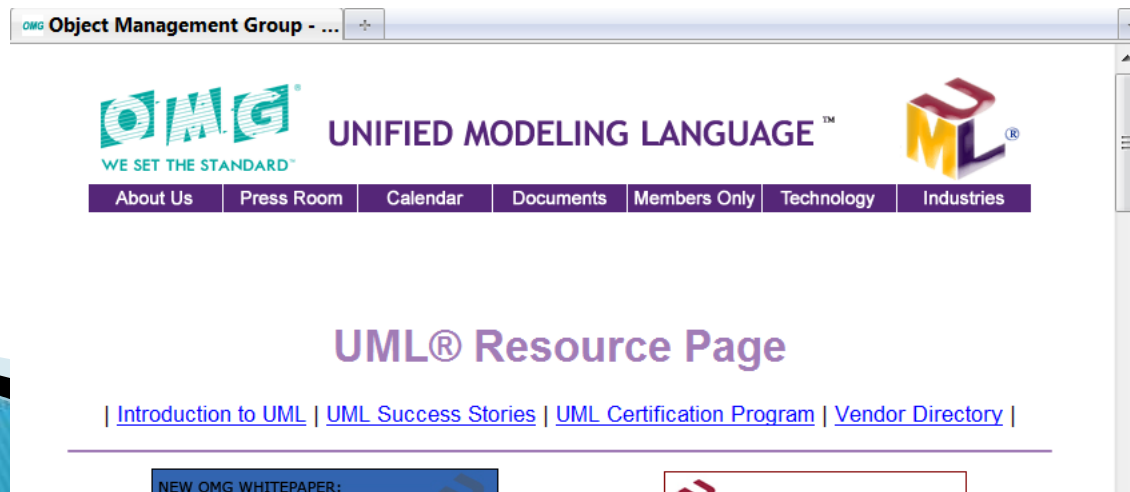
- 1990:** Ada/Booch, Booch '91, Booch '93, OMT, OOSE, OOSE 94, Fusion, RDD, OBA, OOA, OODA, OOSA.
- 1995:** UM 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 1997:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 2000:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 2003:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 2005:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 2007:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.
- 2008:** UML 0.8, UML 0.9, UML 1.1, UML 1.3, UML 1.4, UML 1.5, UML 2.0, UML 2.1.2, UML 2.2, SysML 1.1, BPMN 1.1, DSLs.

UML – Definiție (OMG)

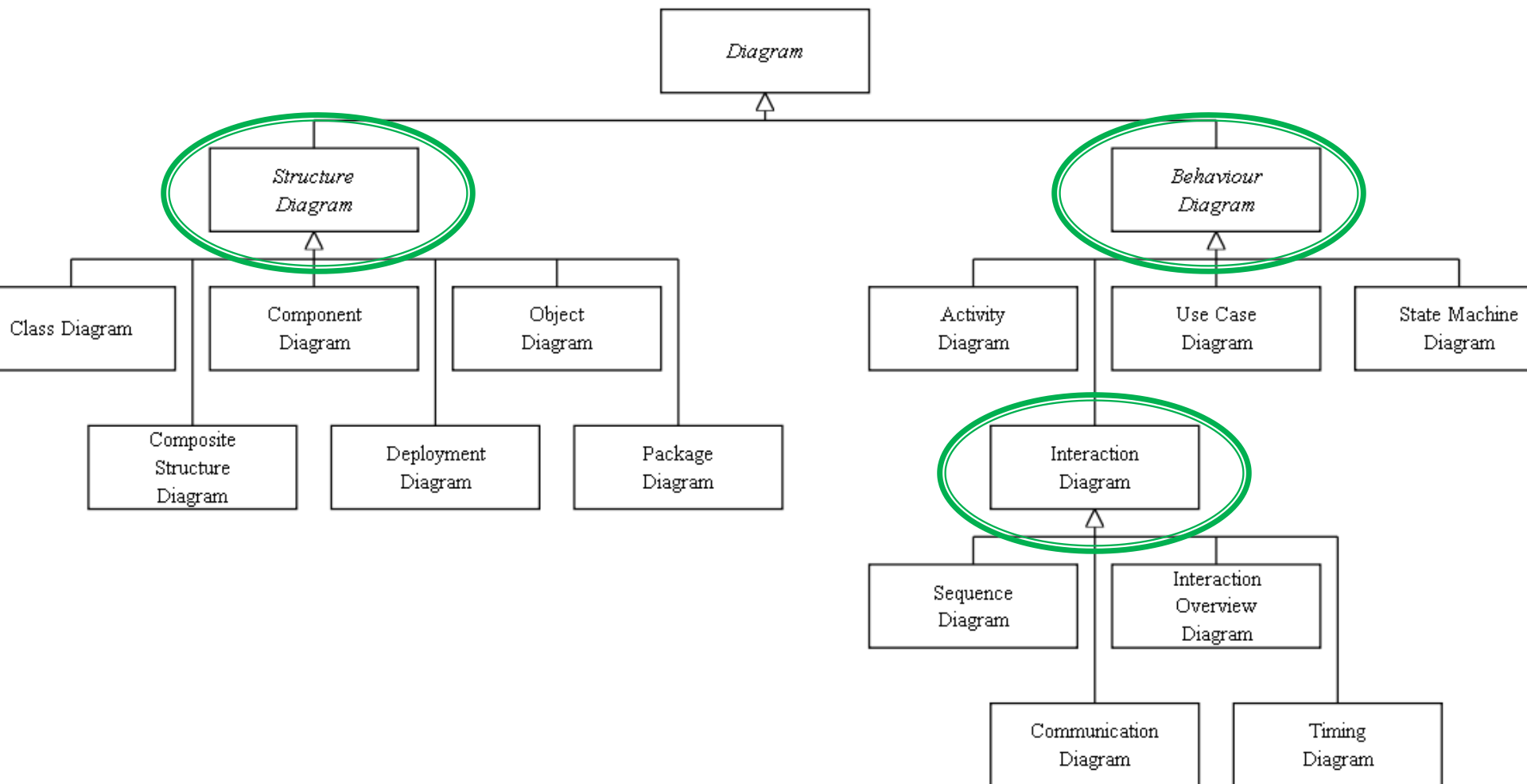
- ▶ *"The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.*
- ▶ *The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components."*

UML – Standard Internațional

- ▶ Ianuarie 1997 – UML 1.0 a fost propus spre standardizare în cadrul OMG (Object Management Group)
- ▶ Noiembrie 1997 – Versiunea UML 1.1 a fost adoptată ca standard de către OMG
- ▶ Ultima versiune este UML 2.4.1
- ▶ Site-ul oficial: <http://www.uml.org>

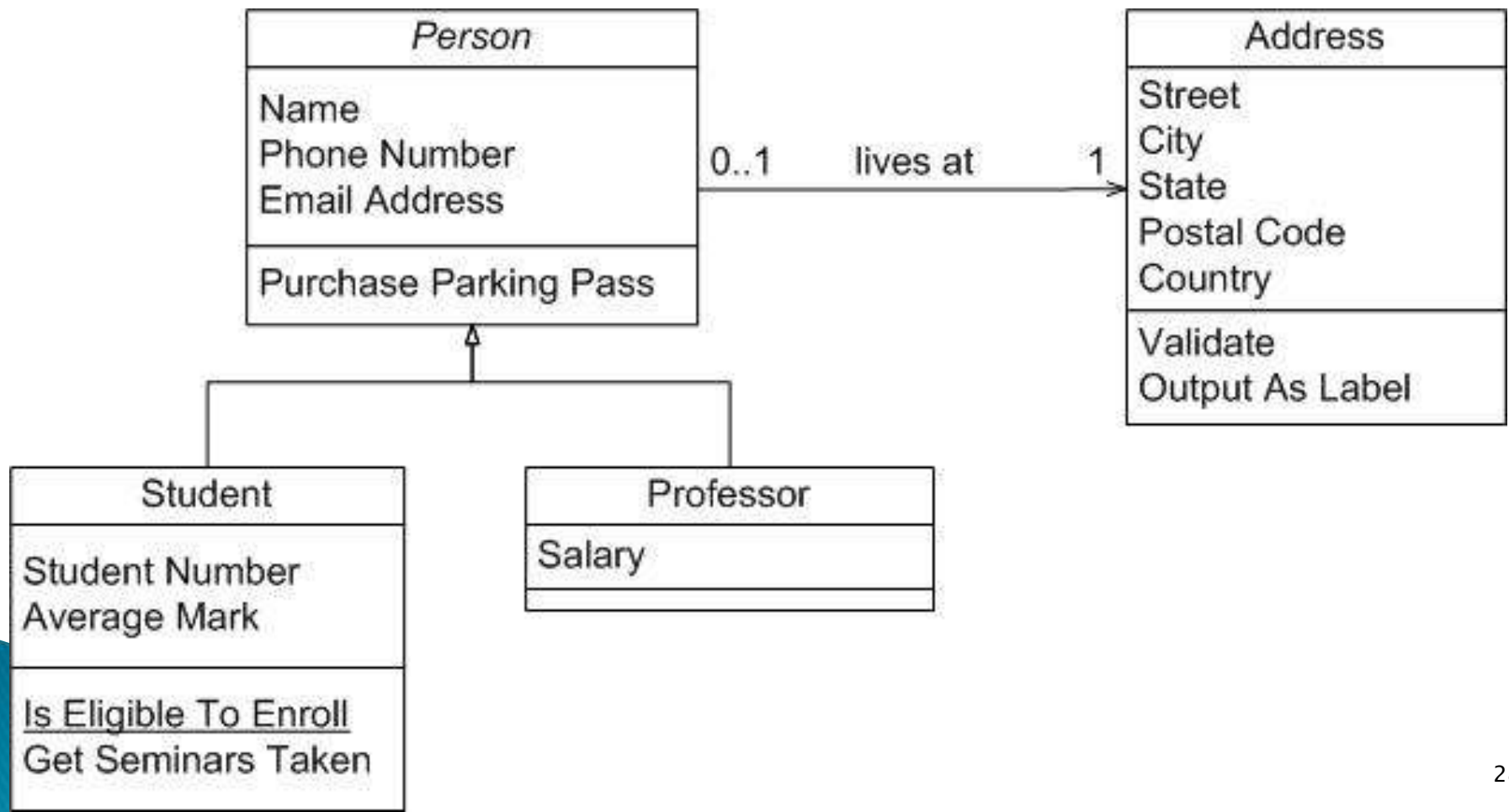


UML2.0 – 13 Tipuri de Diagrame



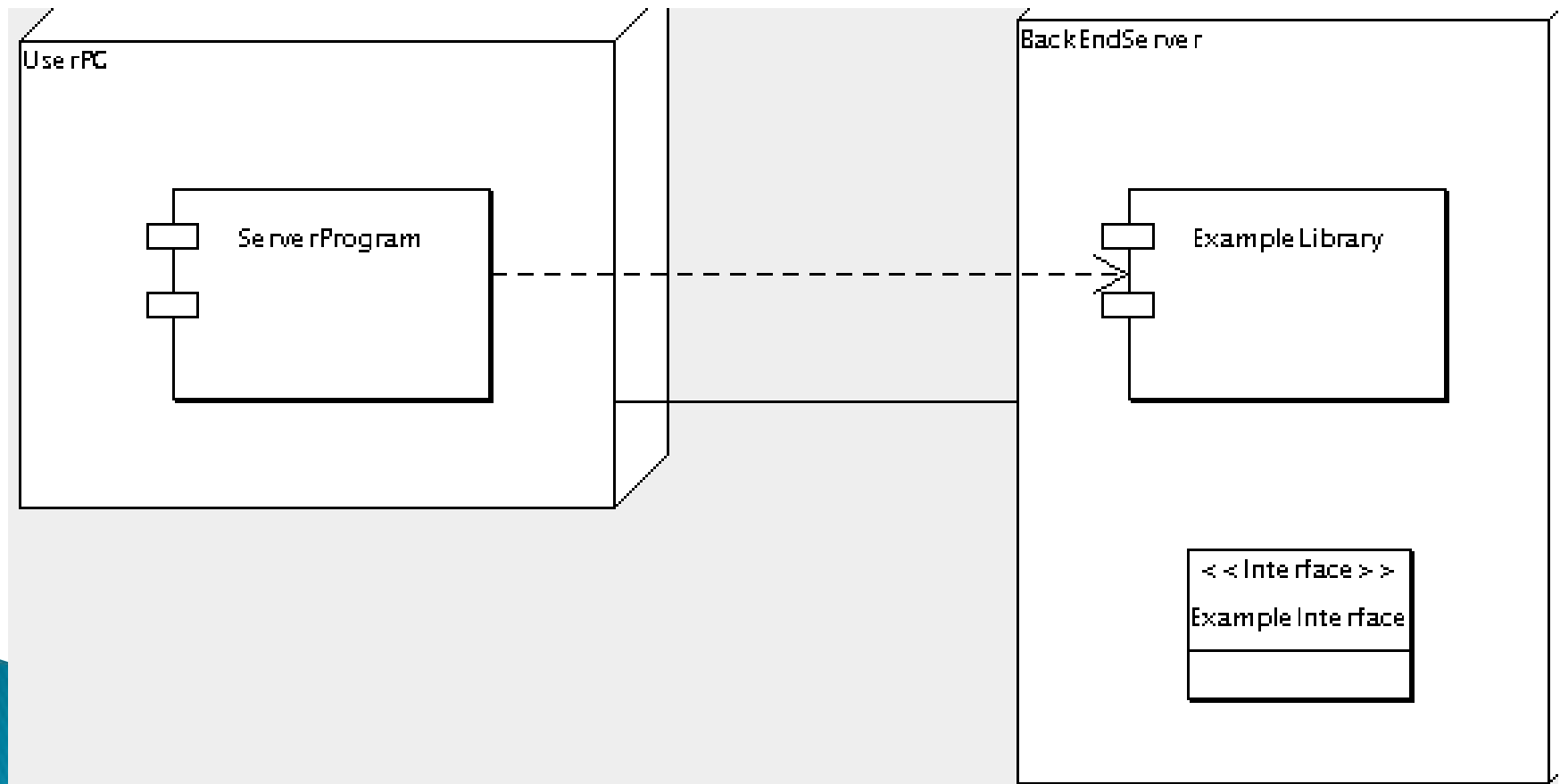
UML2.0 – Diagrame de Structură 1

- ▶ **Diagrame de Clasă:** clasele (atributele, metodele) și relațiile dintre clase



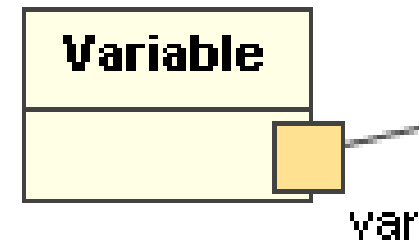
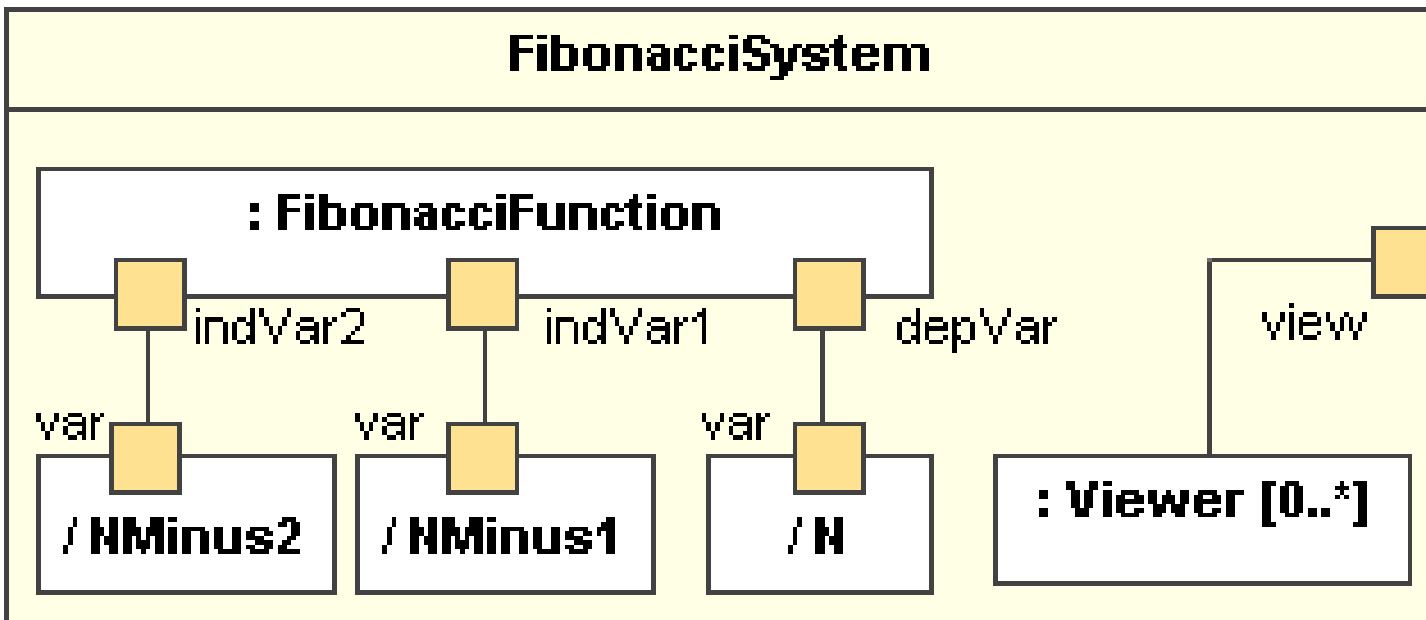
UML2.0 – Diagrame de Structură 2

- ▶ **Diagramă de Componente:** componentele sistemului și legăturile între componente



UML2.0 – Diagrame de Structură 3

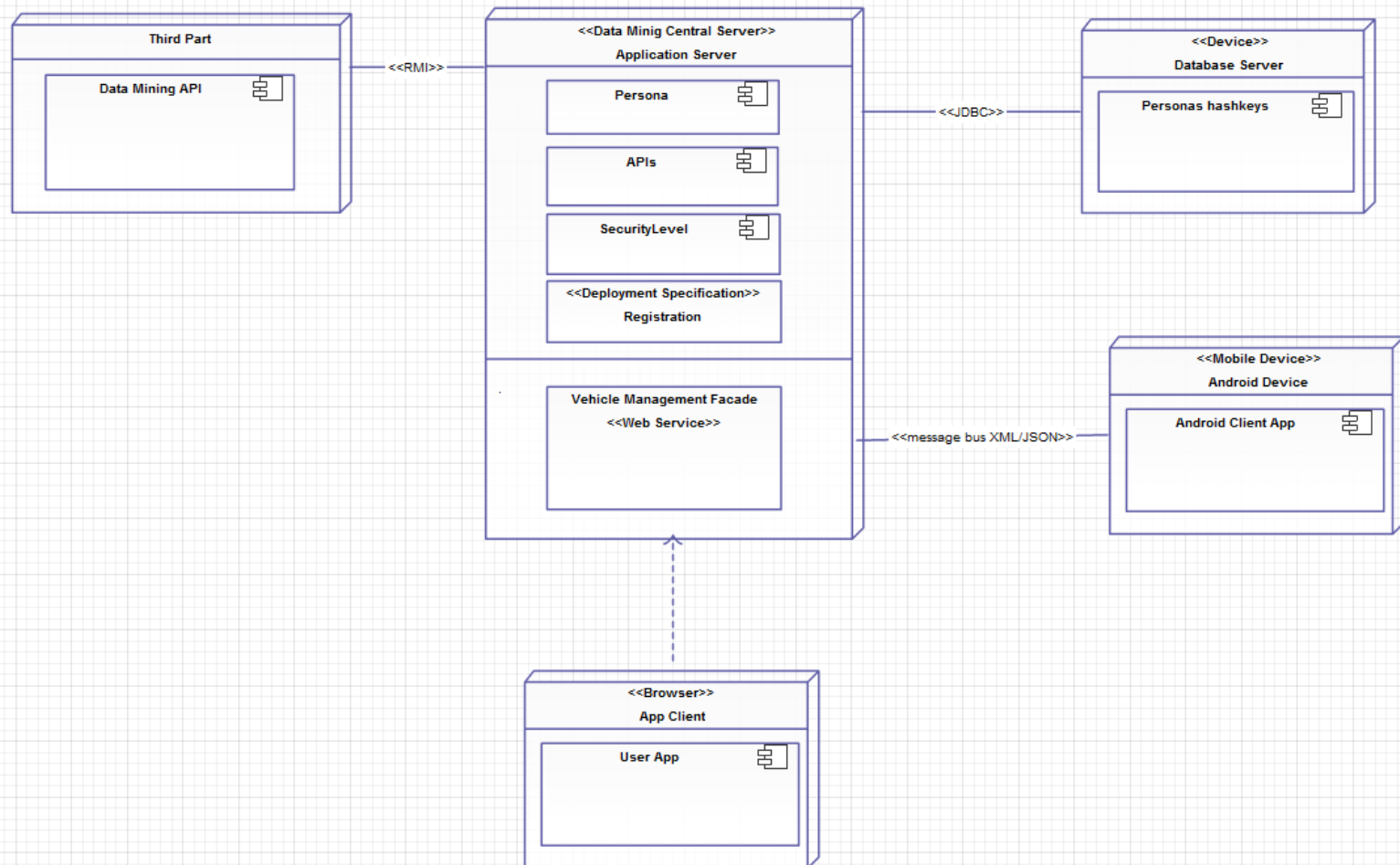
- ▶ Diagrame structură composită: structura internă



UML2.0 – Diagrame de Structură 4

► Diagramă de Deployment: modelarea structurii hardware

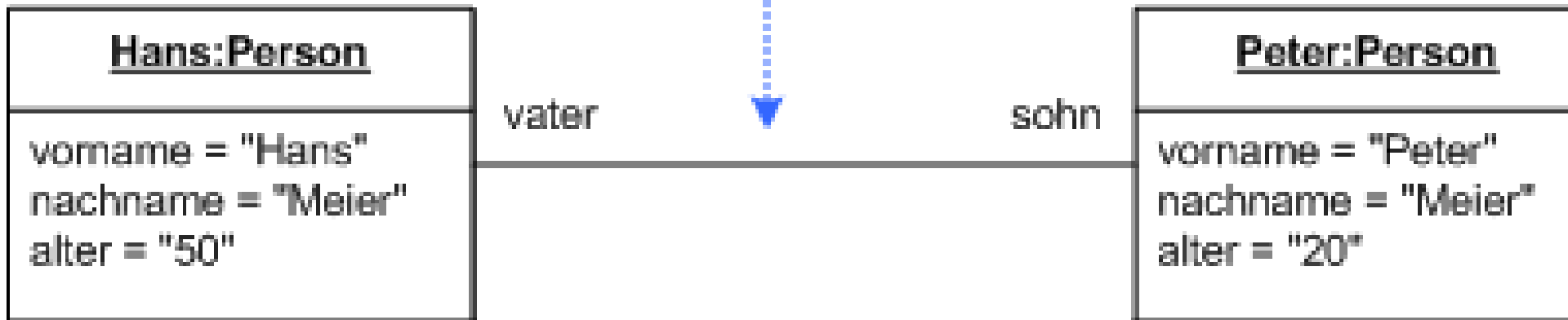
Deployment Diagram For Web DataMining System



UML2.0 – Diagramme de Structură 5

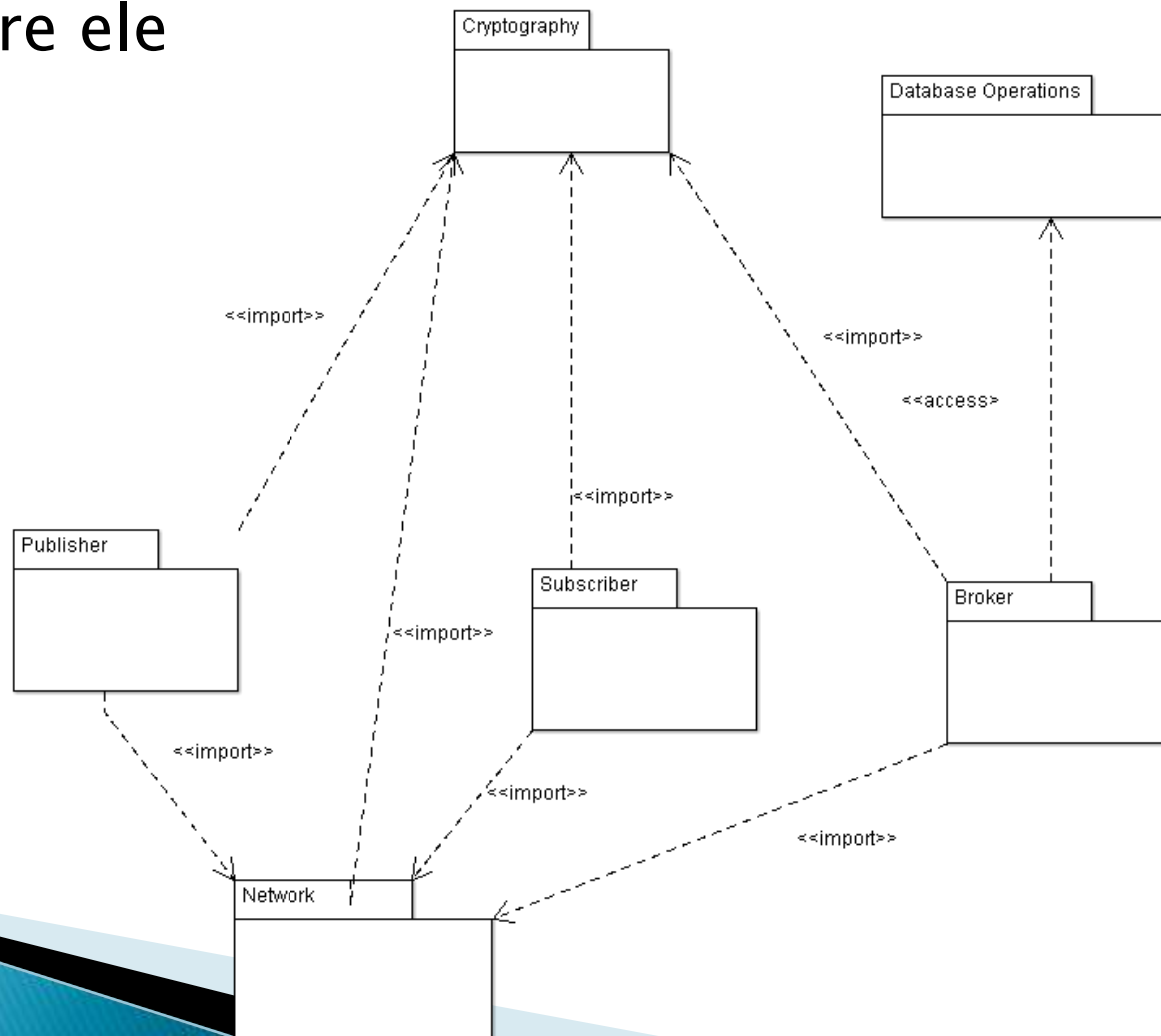
- ▶ **Diagramă de obiecte:** structura sistemului la un moment dat

Ausprägungsspezifikation für eine
Objektbeziehung



UML2.0 – Diagrame de Structură 6

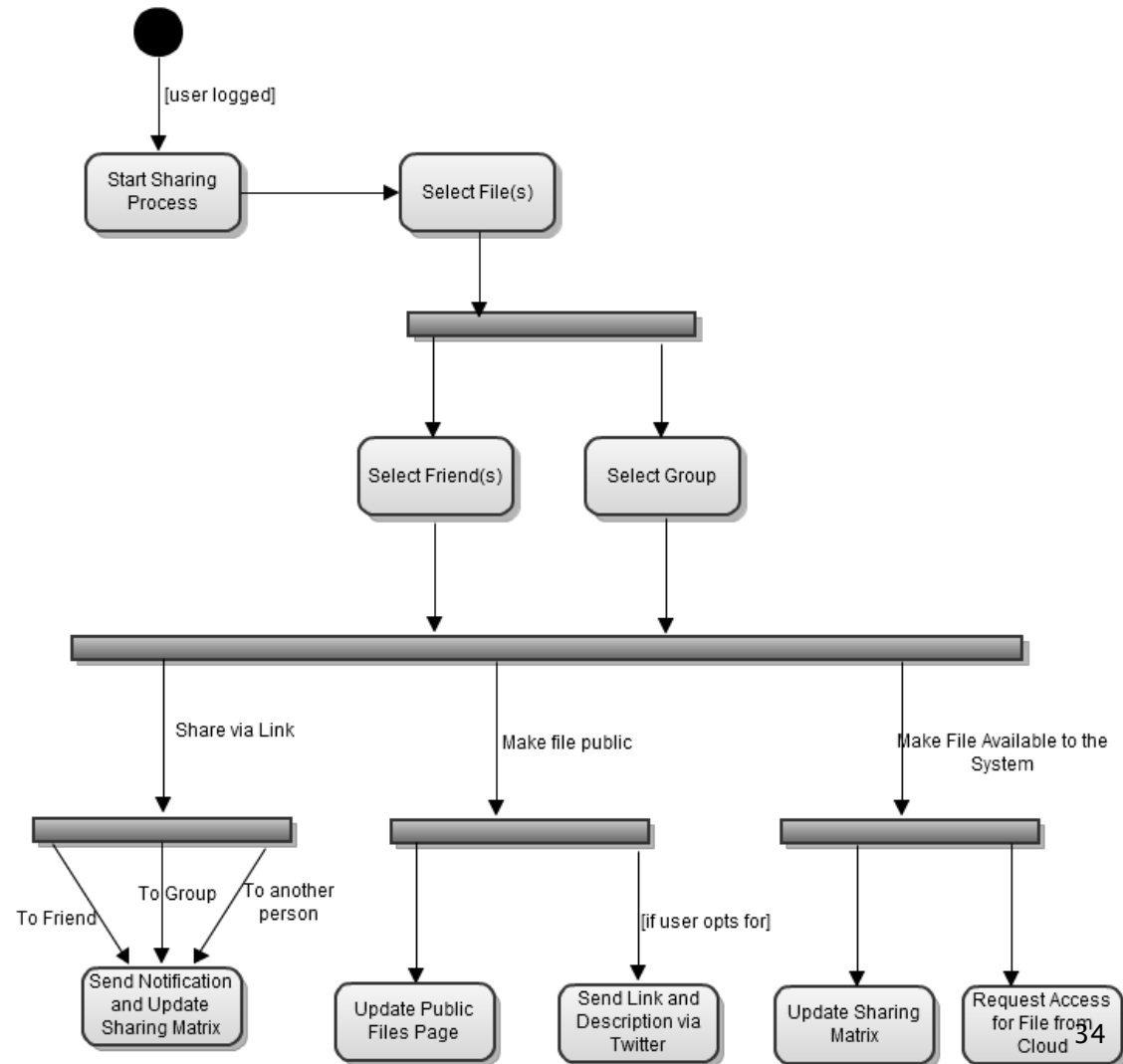
- ▶ **Diagramă de pachete:** împărțirea sistemului în pachete și relațiile dintre ele



UML 2.0 – Diagrame Comportamentale 1

- ▶ **Diagrame de activitate:** prezentare business și a fluxului de activități

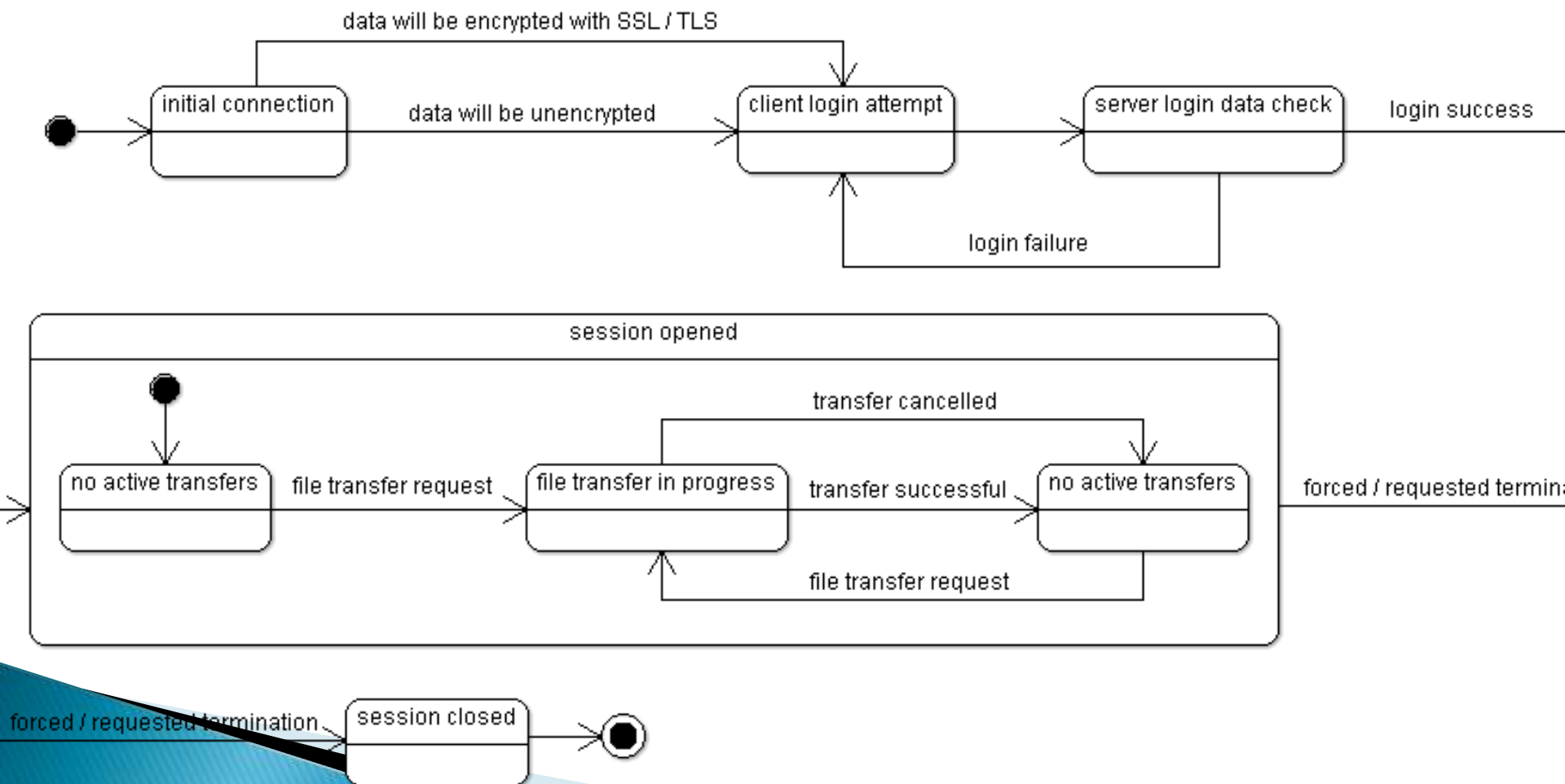
File Sharing: Activity Diagram



UML 2.0 – Diagrame Comportamentale 2

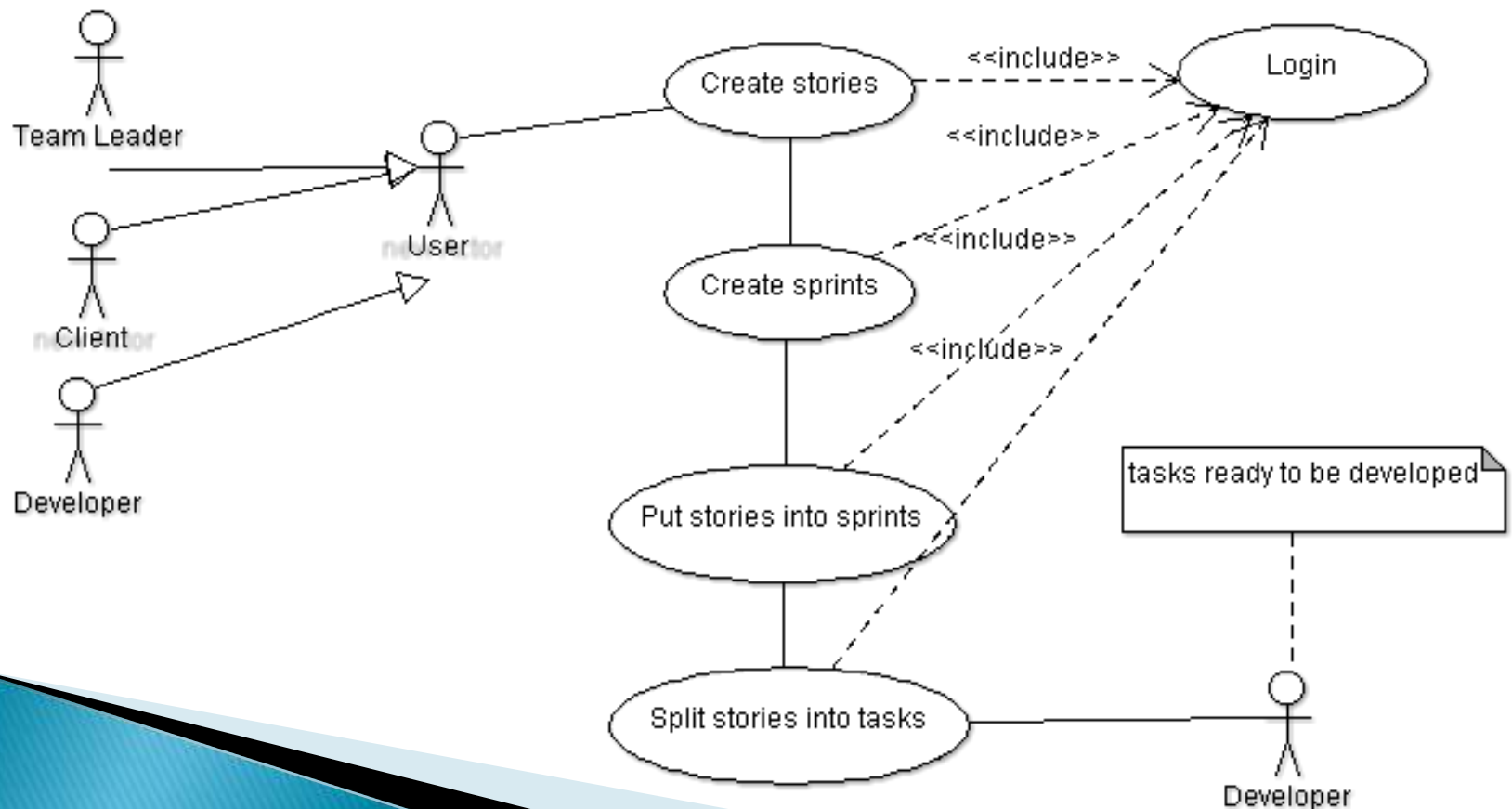
- ▶ **Diagrame de stare:** pentru a prezenta mai multe sisteme

State diagram for the FTP protocol



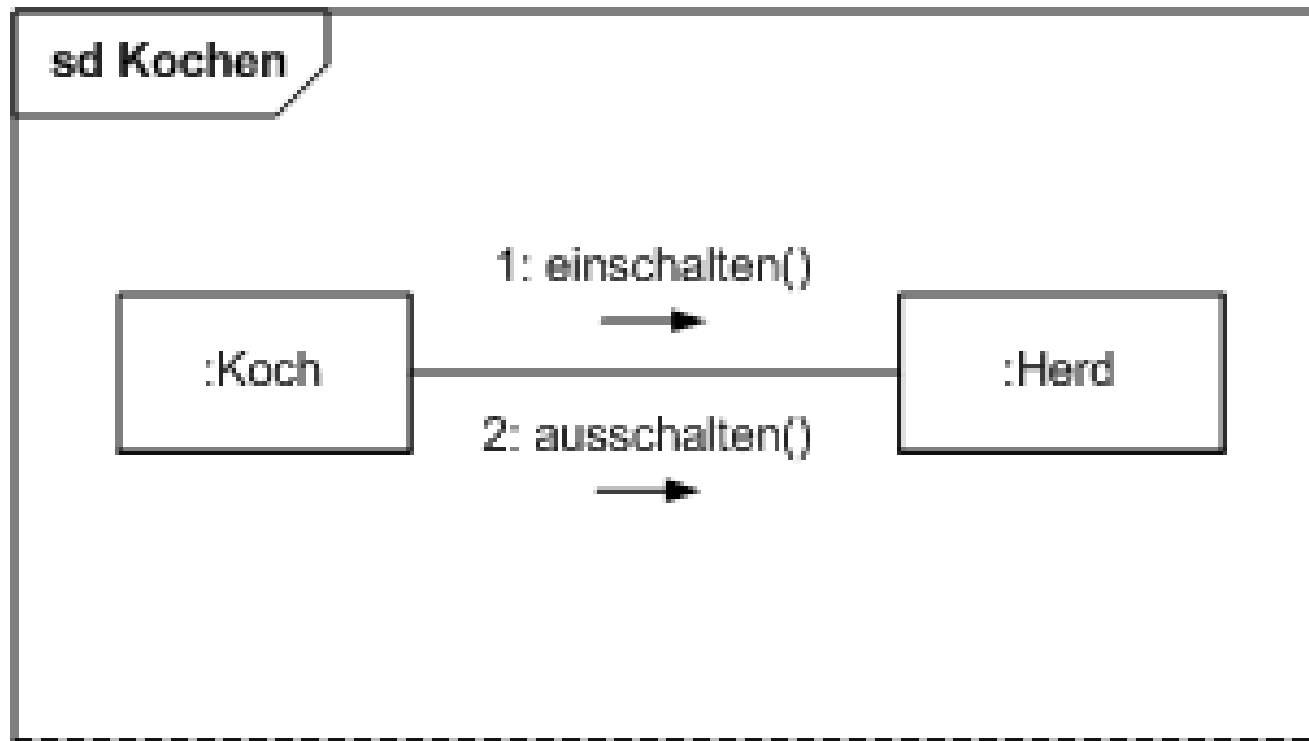
UML 2.0 – Diagrame Comportamentale 3

- **Diagrame Use Case:** prezintă funcționalitățile sistemului folosind actori, use case-uri și dependențe între ele



UML 2.0 – Diagrame de interacțiuni 1

- ▶ **Diagrama de comunicare:** arată interacțiunile între obiecte (comportamentul dinamic al sistemului) (**actori:** bucătar, aragaz, **acțiuni:** gătit, aprinderea, deconectarea)



UML 2.0 – Diagrame de interacțiuni 2

- **Diagramă de secvență:** prezintă modul în care obiectele comunică între ele din punct de vedere al trimiterii de mesaje

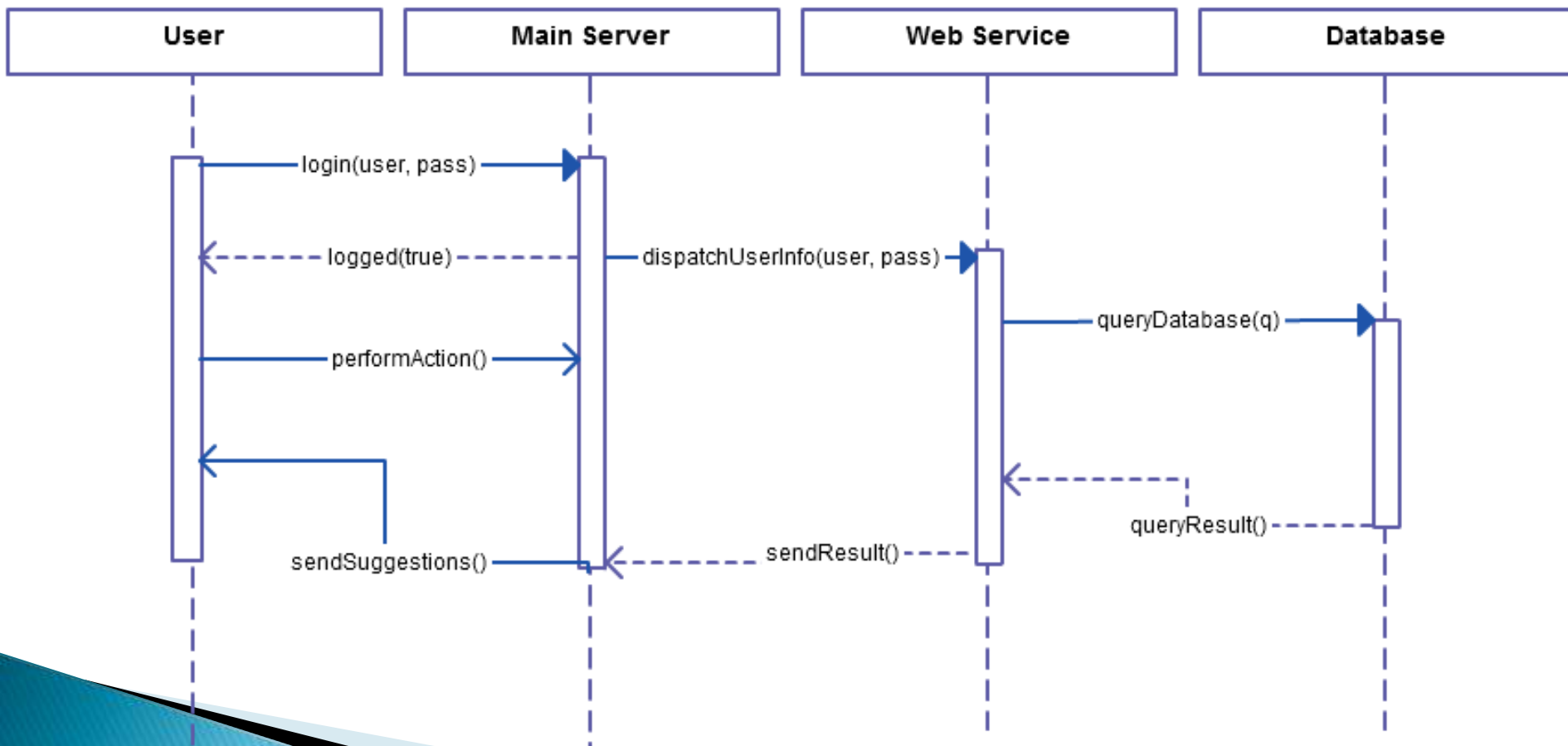


Diagrama cazurilor de utilizare (*Use Case Diagram*)

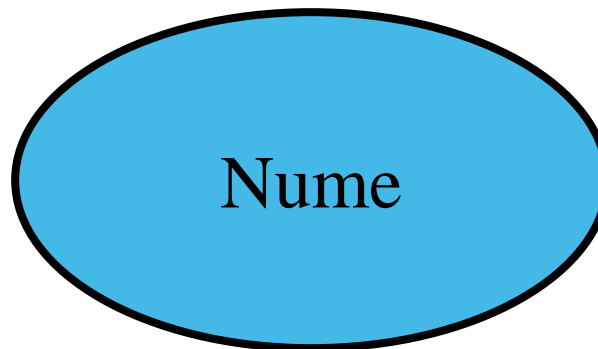
- ▶ Este o diagramă comportamentală care captează cerințele sistemului
- ▶ Delimitează granițele sistemului
- ▶ Punctul de plecare îl constituie scenariile de folosire a sistemului din fișa cerințelor
- ▶ Poate prezenta:
 - specificarea cerințelor (externe) din punctul de vedere al utilizatorului
 - specificarea funcționalității sistemului din punctul de vedere al sistemului
- ▶ Conține:
 - **UseCase**–uri = funcționalități ale sistemului
 - **Actori** = entități externe cu care sistemul interacționează
 - **Relații**

UseCase

- ▶ Este o descriere a unei mulțimi de secvențe de acțiuni (incluzând variante) pe care un program le execută atunci când interacționează cu entitățile din afara lui (*actori*) și care conduc la obținerea unui rezultat observabil
- ▶ Poate fi un sistem, un subsistem, o clasă, o metodă
- ▶ Reprezintă o funcționalitate a programului
- ▶ Precizează ce face un program sau subprogram
- ▶ Nu precizează cum se implementează o funcționalitate
- ▶ Identificarea UseCase-urilor se face pornind de la cerințele clientului și analizând descrierea problemei.

UseCase – Reprezentare

- ▶ Notăție
- ▶ Atribute
 - Nume = **fraza verbală ce denumește o operație sau un comportament** din domeniul problemei.
- ▶ Restricții
 - Numele este unic

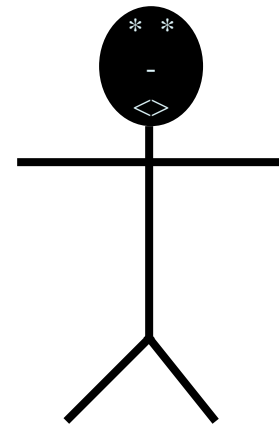


Actor

- ▶ Reprezintă **un rol** pe care utilizatorii unui UseCase îl joacă atunci când interacționează cu acesta
- ▶ Este o entitate exterioară sistemului
- ▶ Interacționează cu sistemul:
 - Inițiază execuția unor cazuri de utilizare
 - Oferă funcționalitate pentru realizarea unor cazuri de utilizare
- ▶ Poate fi:
 - Utilizator (uman)
 - Sistem software
 - Sistem hardware

Actor – Reprezentare


- ▶ Notăție
- ▶ Atribute
- ▶ Nume = **indică rolul** pe care actorul îl joacă în interacțiunea cu un UseCase
- ▶ Restricții
 - Numele este unic




Relații

- ▶ Se stabilesc între două elemente
- ▶ Tipuri de relații:
 - **Asociere:** Actor – UseCase, UseCase – UseCase
 - **Generalizare:** Actor – Actor, UseCase – UseCase
 - **Dependență:** UseCase – UseCase (<<include>>, <<extend>>)

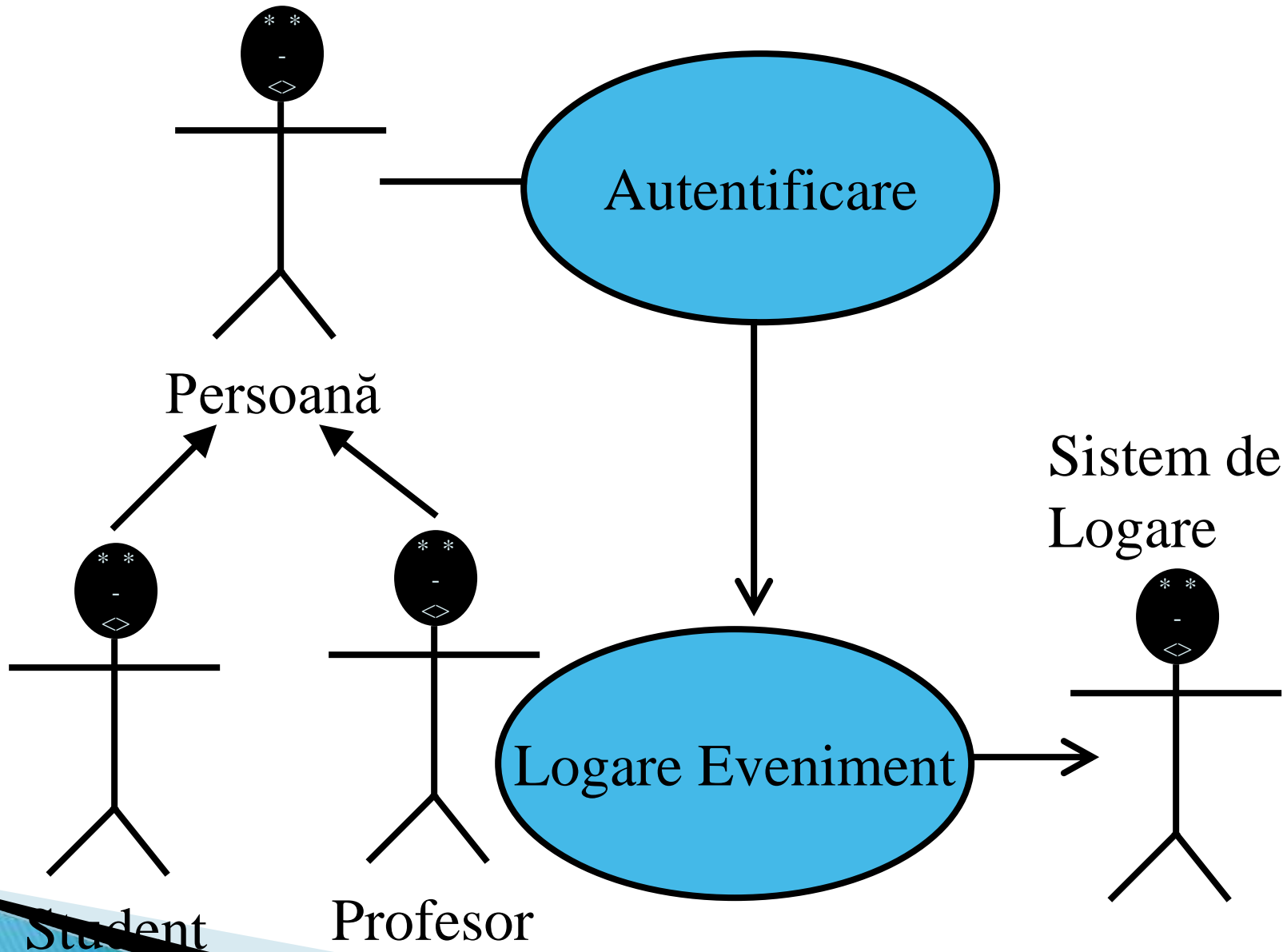
Relația de Asociere

- ▶ Modelează o comunicare între elementele pe care le conectează
- ▶ Poate sa apară între
 - **un actor și un UseCase** (actorul inițiază execuția cazului de utilizare sau oferă funcționalitate pentru realizarea acestuia)
 - **două UseCase-uri** (transfer de date, trimitere de mesaje/semnale)
- ▶ **Notăție** 

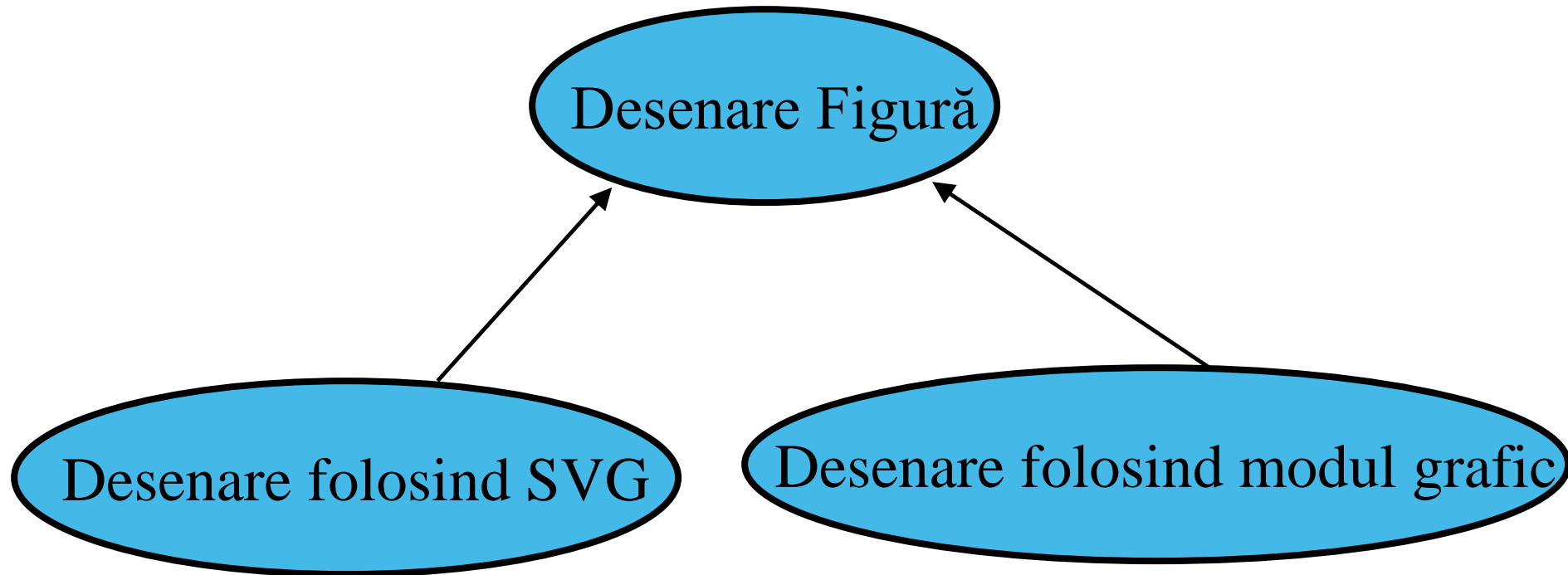
Relația de Generalizare

- ▶ Se realizează între elemente de același tip \Rightarrow ierarhii
- ▶ Modelează situații în care un element este un caz particular al altui element
- ▶ Elementul particular moștenește relațiile în care este implicat elementul general
- ▶ Notăție: 

Exemplu 1



Relația de Generalizare



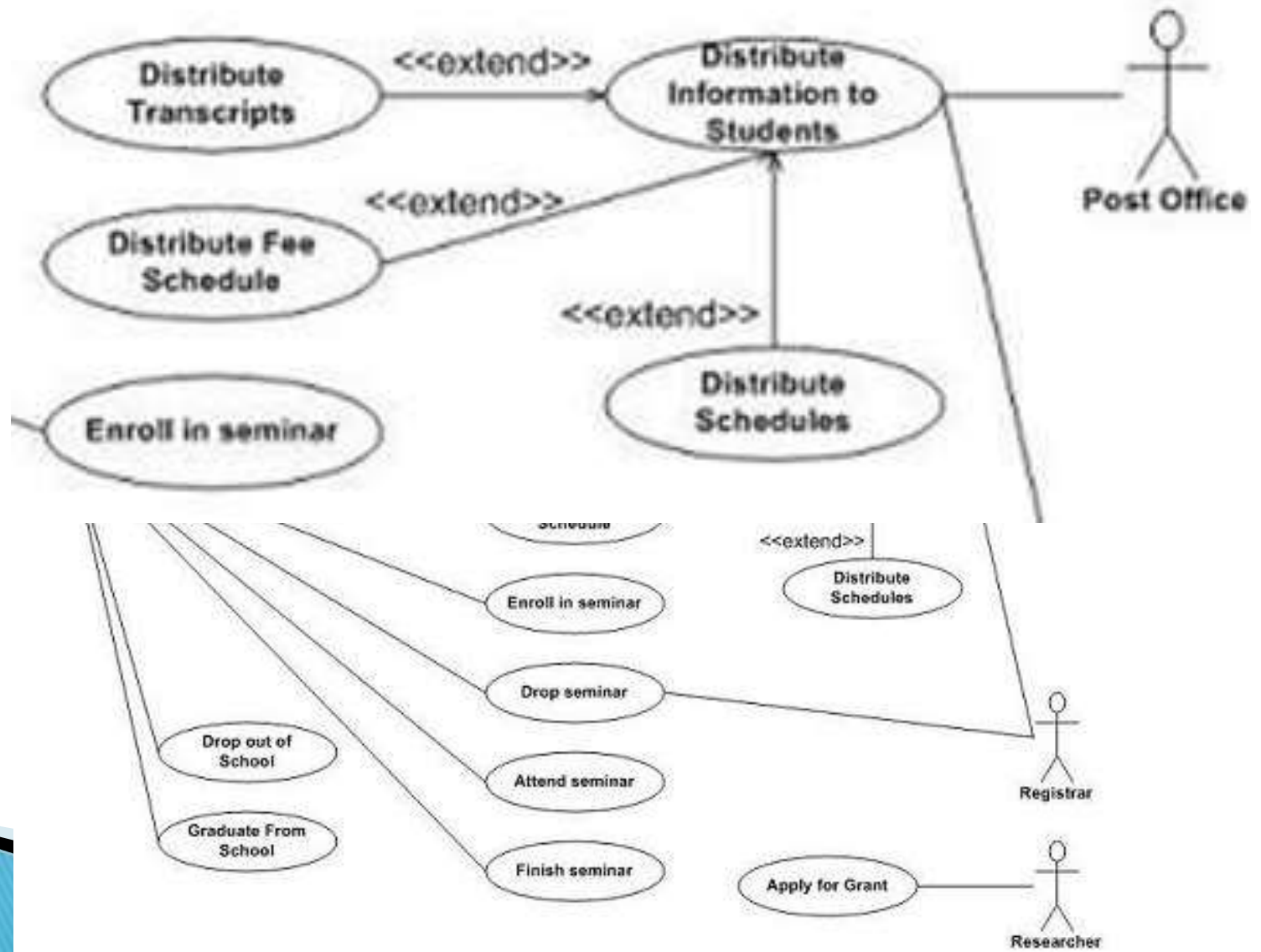
Relația de Dependență

- ▶ Apare între două UseCase-uri
- ▶ Modelează situațiile în care
 - Un UseCase **folosește** comportamentul definit în alt UseCase (<<include>>)
 - Comportamentul unui UseCase **poate fi extins** de către un alt UseCase (<<extend>>)

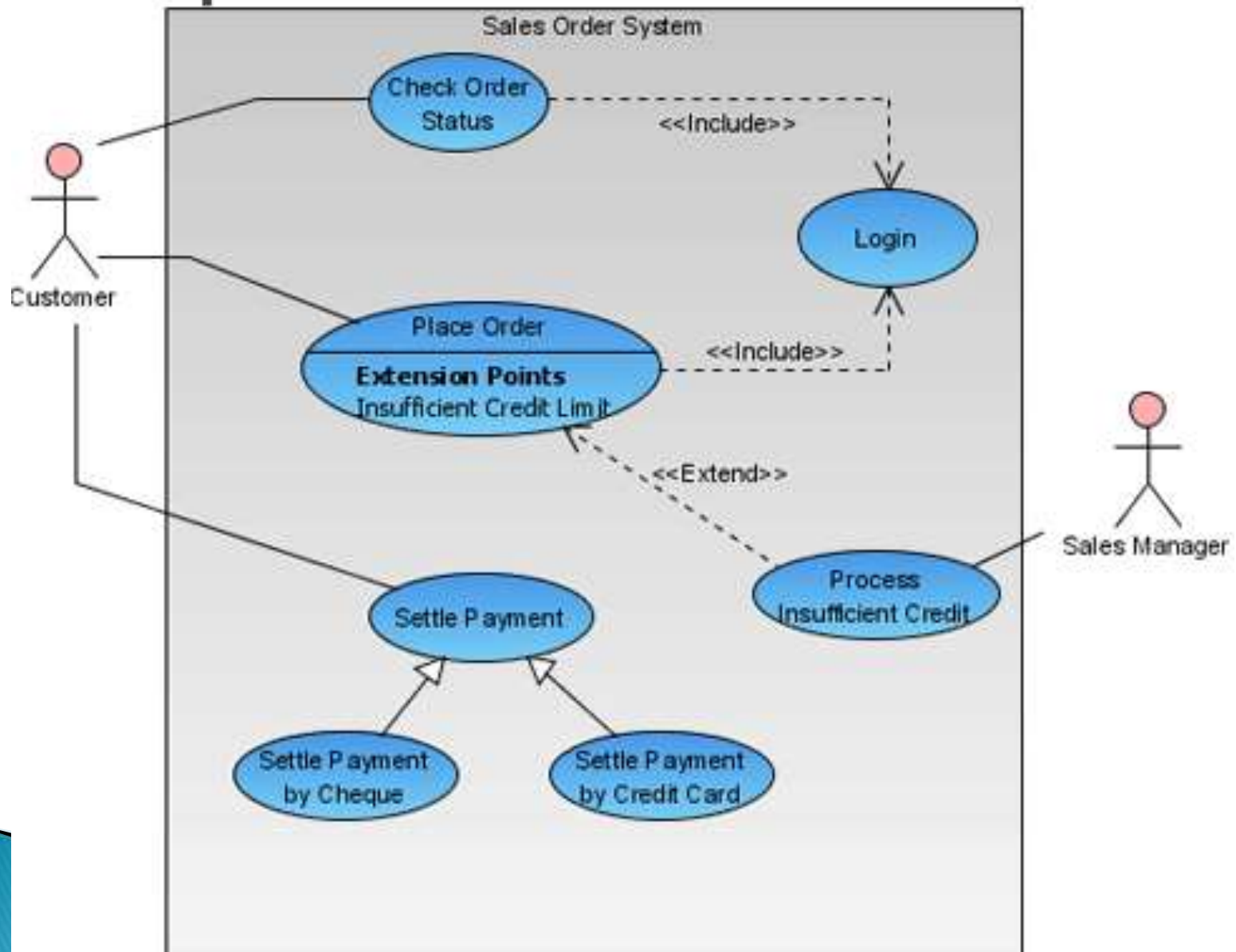
▶ Notăție



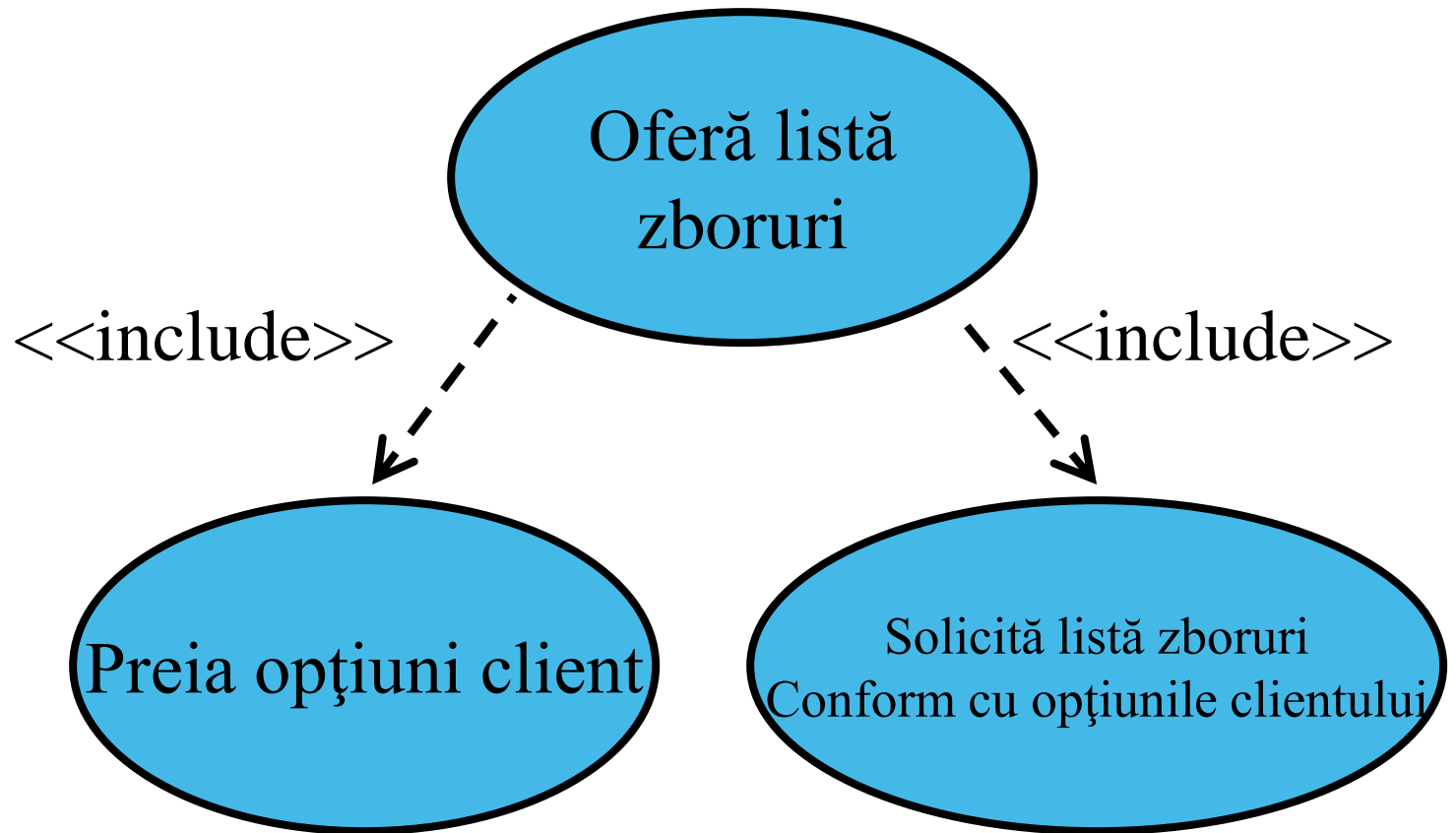
Exemplul 2



Exemplul 3



Exemplu 4



Concepte OO – Recapitulare

▶ **Obiect, Clasă, Instanță**

▶ **Obiect:** Entitate care are: identitate, stare, comportament

- Exemplu: Mingea mea galbena de tenis, cu diametrul de 10 cm, care sare

▶ **Clasă:** Descriere a unei mulțimi de obiecte cu aceleași caracteristici structurale, aceleași caracteristici comportamentale

- Exemplu: mingi care au culoare, diametru, întrebuințare, sar

▶ **Instanță:** un obiect care aparține unei clase

- Exemplu: Popescu Viorel este un Student

Orientat Obiect 1

- ▶ Este orice abordare ce cuprinde
 - Încapsularea datelor
 - Moștenire
 - Polimorfism
- ▶ **Încapsularea datelor** (exemplu clasa Punct)
 - Înseamnă punerea la un loc a datelor (atributelor) și a codului (metodelor)
 - Datele pot modificate (doar) prin intermediul metodelor
 - Data hiding: nu ne interesează cum se oferă serviciile, ci doar ca se oferă
 - Dacă se schimbă structura, sau modul de realizare, interfața rămâne neschimbată

Orientat Obiect 2

► Moștenire:

- Anumite clase sunt specializări (particularizări) ale altor clase
- O subclasa are (moștenește) caracteristicile super-clasei, pe care le poate extinde într-un anumit fel
- O instanță a unei clase derivate este în mod automat și o instanță a clasei de bază
- Exemplu (Student – Persoana)

► Polimorfism

- Interpretarea semanticii unui apel de metoda se face de către cel care primește apelul
- Exemplu: Eu spun unei forme: DESENEAZĂ-TE. Ea, dacă e pătrat trage 4 linii, dacă e cerc, face niște puncte de jur împrejurul centrului
- De altfel, nu mă interesează cine, cum face

Diagrama de clase – Class Diagram

► Scop:

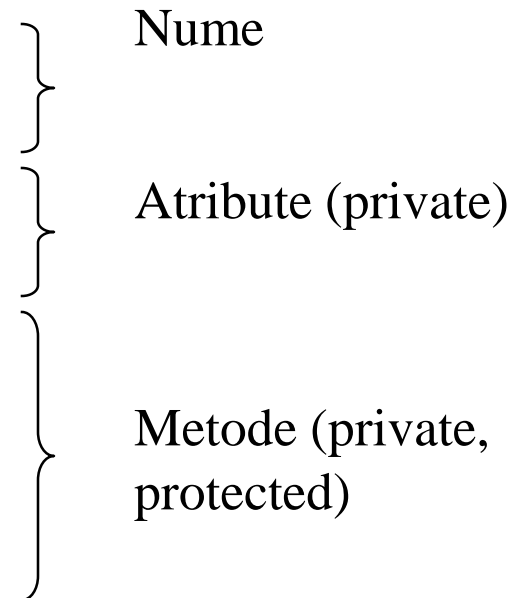
- Modelează vocabularul sistemului ce trebuie dezvoltat
- Surprinde conexiunile semantice sau interacțiunile care se stabilesc între elementele componente
- Folosită pentru a modela structura unui program

► Conține

- Clase/Interfețe
- Obiecte
- Relații (Asocierie, Agregare, Generalizare, Dependență)

Clase

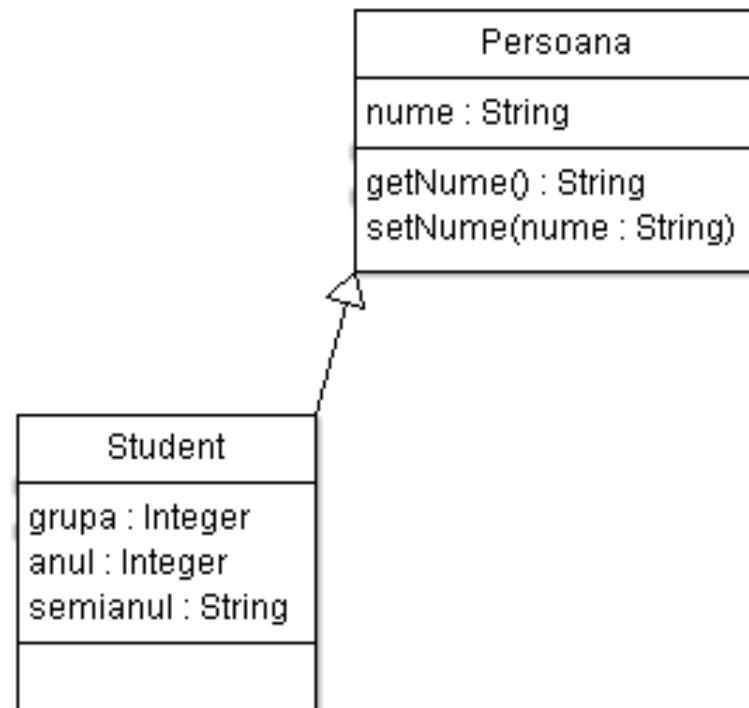
- ▶ Modelează vocabularul = identifică conceptele pe care clientul sau programatorul le folosește pentru a descrie soluția problemei
- ▶ Elementele unei clase:
 - Nume: identica o clasa
 - Atribute: proprietăți ale clasei
 - Metode: implementarea unui serviciu care poate cerut oricărei instanțe a clasei



Relații – Generalizare – C#

- ▶ Modelează conceptul de moștenire între clase
- ▶ Mai poartă denumirea de relație de tip *is a* (este un/este o)

ArgoUML – Relația de generalizare



Relații – Asociere

- ▶ Exprimă o conexiune semantică sau o interacțiune între obiecte aparținând diferitelor clase
- ▶ Pe măsura ce sistemul evoluează noi legături între obiecte pot fi create, sau legături existente pot fi distruse
- ▶ O asociere interacționează cu obiectele sale prin intermediul capetelor de asociere
- ▶ Elemente:
 - Nume: descrie relația
 - Capete de asociere
 - Nume = rolul jucat de obiect în relație
 - Multiplicitate = câte instanțe ale unei clase corespund unei singure instanțe ale celeilalte clase



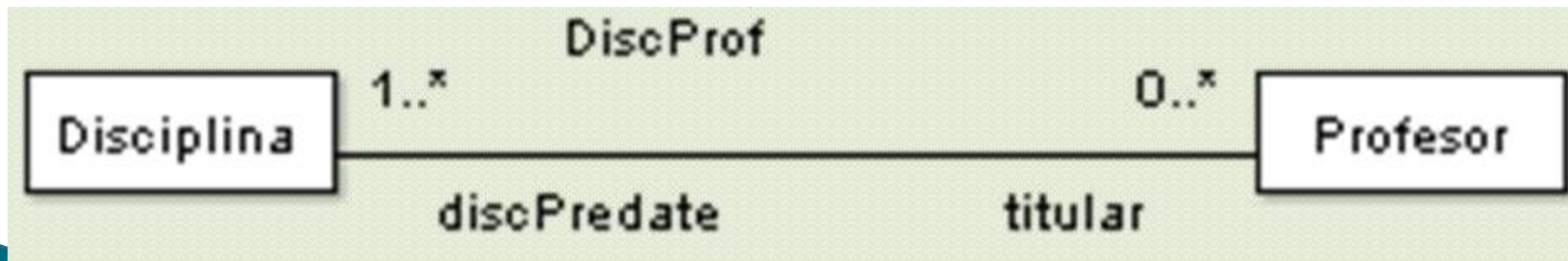
Relația de asociere 1

- ▶ Relația Student – Disciplină
 - **Student**: urmez 0 sau mai multe discipline, cunosc disciplinele pe care le urmez;
 - **Disciplină**: pot fi urmată de mai mulți studenți, nu cunosc studenții care mă urmează



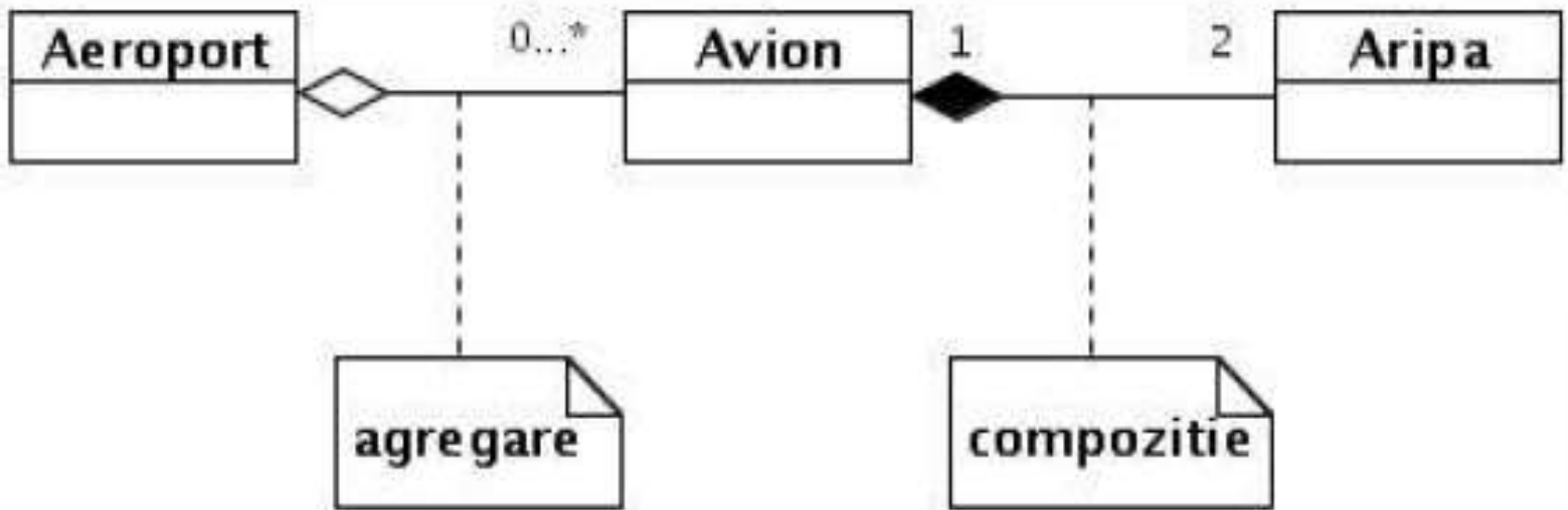
Relația de asociere 2

- ▶ Relația Disciplină – Profesor
 - **Disciplină**: sunt predată de un profesor, îmi cunosc titularul
 - **Profesor**: pot preda mai multe discipline, cunosc disciplinele pe care le predau

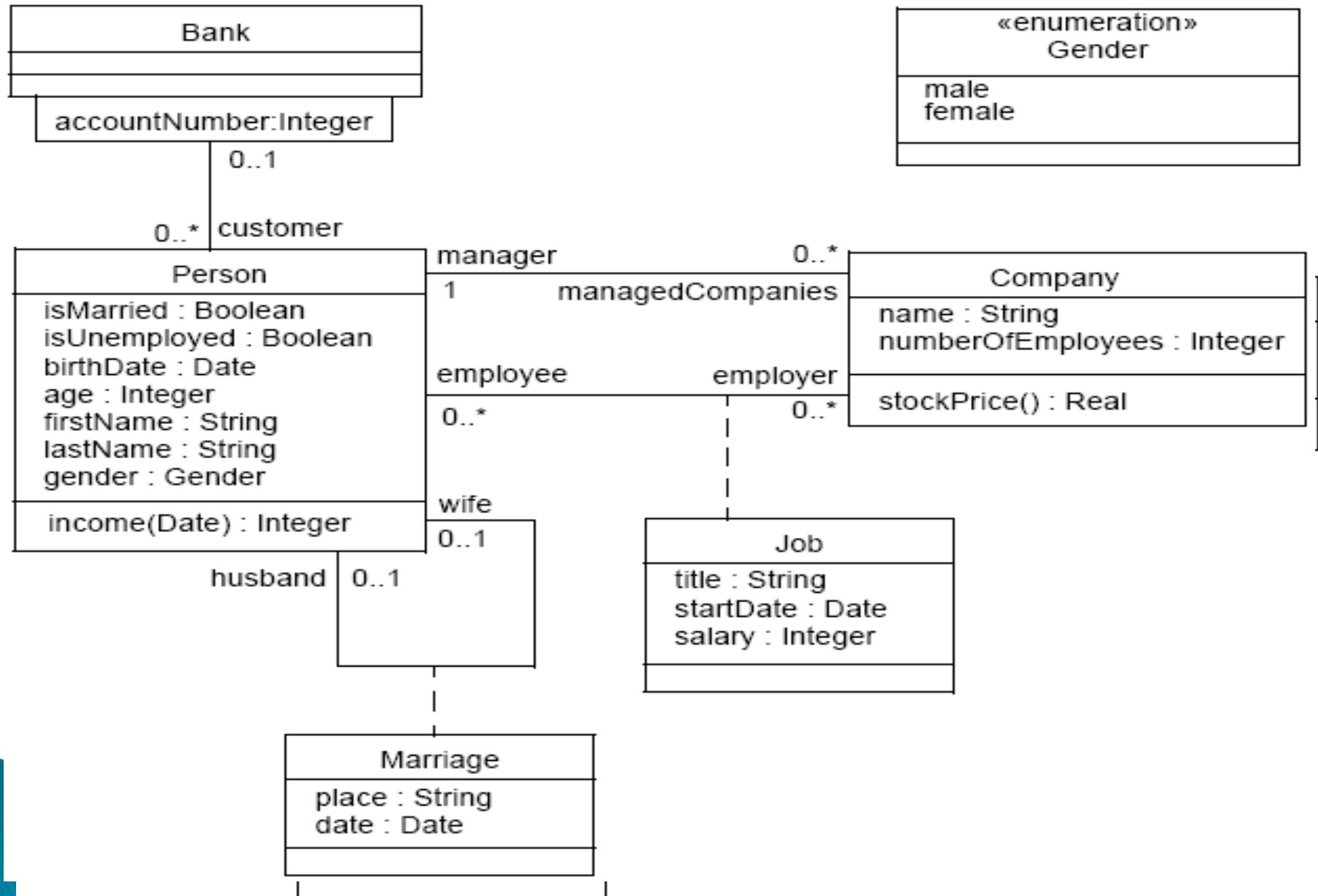


Relații – Agregare

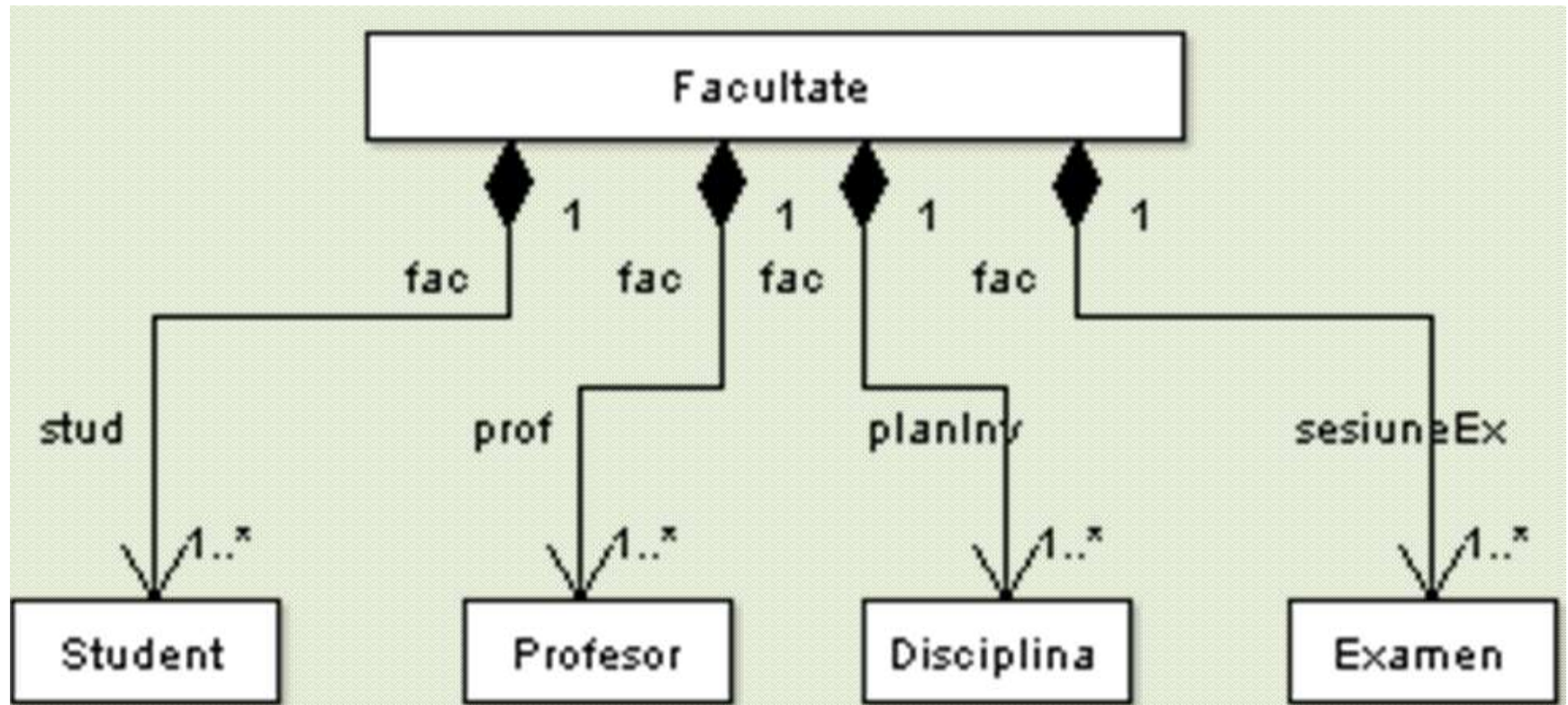
- ▶ Este un caz particular al relației de asociere
- ▶ Modelează o relație de tip parte-întreg
- ▶ Poate avea toate elementele unei relații de asociere, însă în general se specifică numai multiplicitatea
- ▶ Se folosește pentru a modela situațiile între care un obiect este format din mai multe componente.



Exemplul



Relația de Compoziție (“hasA”)



Studiu de Caz

- ▶ Obținerea Studenților Bursieri
 - Actori
 - Scenarii de utilizare

ArgoUML

- ▶ Link: <http://argouml-downloads.tigris.org/argouml-0.34/>
- ▶ Varianta “zip” trebuie doar dezarhivată
- ▶ Trebuie să aveți instalat Java
 - În Path să aveți c:\Program Files\Java\jdk1.6.0_03\bin
 - Variabila JAVA_HOME=c:\Program Files\Java\jdk1.6.0_03\

Concluzii

- ▶ Modelare – De ce?
- ▶ Limbaje grafice
- ▶ UML
 - Structurale: clase
 - Comportamentale: use-case
 - De interacțiuni

Bibliografie

- ▶ **OMG Unified Modeling Language™ (OMG UML), Infrastructure, Version 2.2, May 2008,** <http://www.omg.org/docs/ptc/08-05-04.pdf>
- ▶ **ArgoUML User Manual, A tutorial and reference description,** <http://argouml-stats.tigris.org/documentation/printablehtml/manual/argomanual.html>
- ▶ **Ovidiu Gheorghieș, Curs IP, Cursurile 3, 4**
- ▶ **Diagrame UML, Regie.ro**

Links

- ▶ OOSE: <http://cs-exhibitions.uni-klu.ac.at/index.php?id=448>
- ▶ ArgoUML: <http://argouml-stats.tigris.org/nonav/documentation/manual-0.22/>
- ▶ Wikipedia