

M. Caramihai, © 2020

**PROGRAMAREA
ORIENTATA
OBIECT**

CURS 5

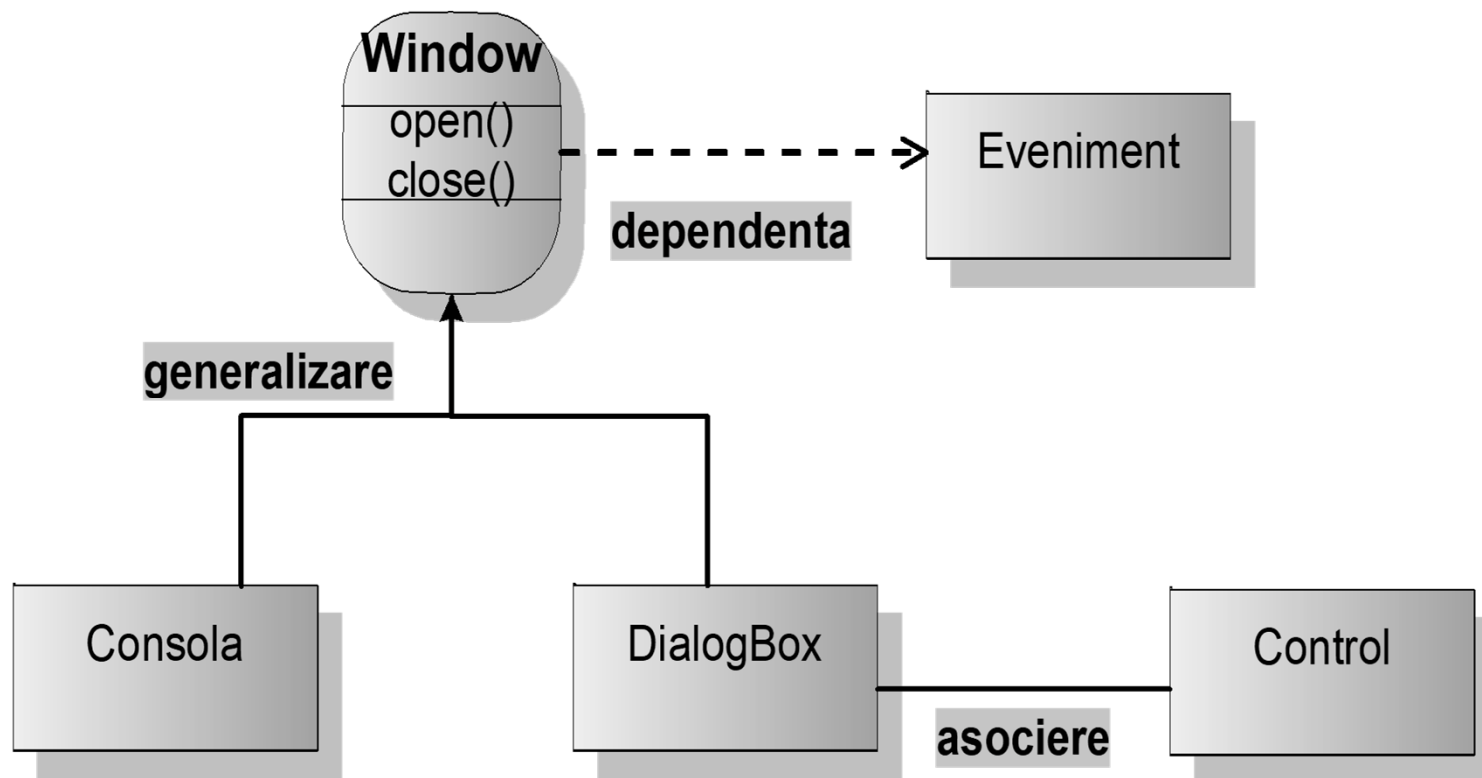
Introducere in UML (2)



Relatii (1)

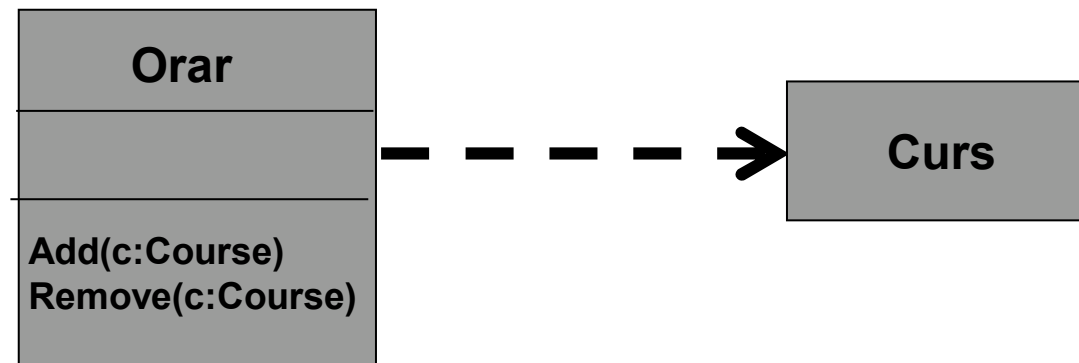
- Relatiile ofera o structura de comunicare intre obiecte
- Diagramele de secvente / colaborari sunt examinate pentru a determina ce legaturi trebuiesc stabilite intre obiecte pentru a permite evolutia sistemului; daca doua obiecte "vorbesc", o legatura (*link*) trebuie sa existe intre ele.
- Tipuri 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

Relatii (2)



Dependenta (1)

- 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 (2)

- 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 – implementare in C++

```
class A { ... };
```

```
class B
```

```
{public:void  
    Executa1(void);};
```

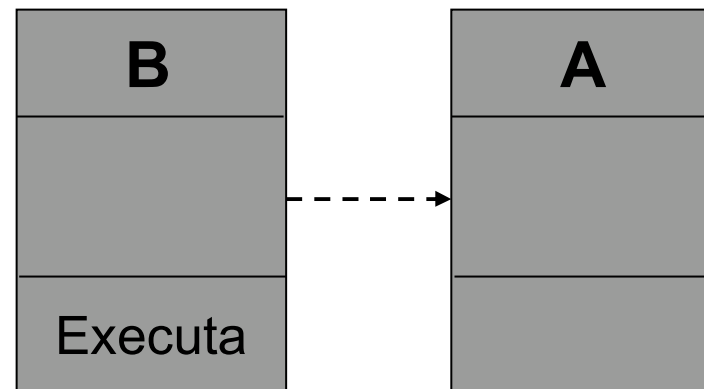
```
void B::Executa1(void)
```

```
{A a;
```


```
a * a1 = new A;
```

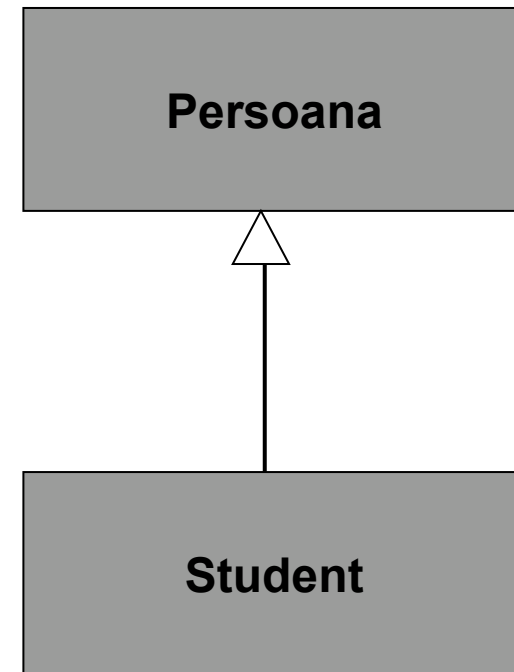
```
// ...
```

```
delete a1;}
```



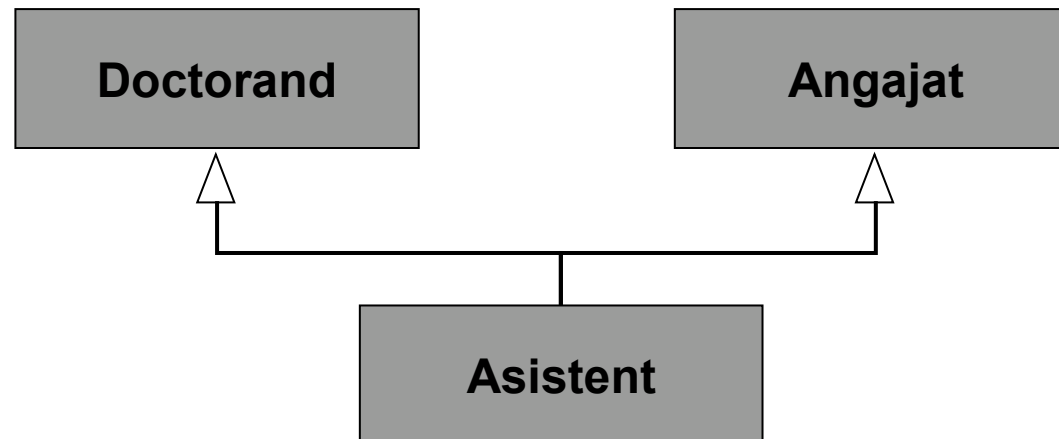
Generalizare (1)

- 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 



Generalizare (2)

- UML permite ca o clasa sa mosteneasca mai multe superclase.
- Observatie: anumite limbaje OO (d.e. Java) nu permit mostenirea multipla



Generalizare (3)

● Mostenire multipla

➤ Avantaje:

- Structurarea elementelor in diferite forme si cu diferite legaturi (ca in lumea reala)
- Posibilitati multiple de utilizare a atributelor si operatiilor din clasele parinte

➤ Dezavantaje

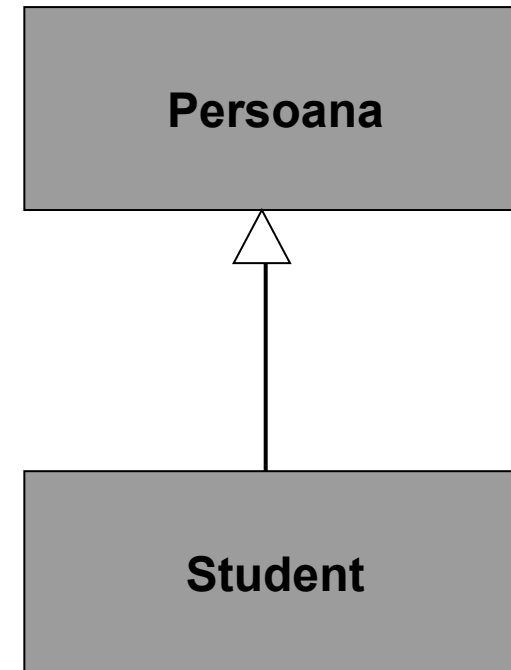
- Orice modificare la nivelul superclasei duce la modificari si in subclasa
- Cand o subclasa mosteneste aceleasi attribute / operatii de la o superclasa, trebuie selectat cu grija ce va fi utilizat

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

Generalizare – implementare in C++

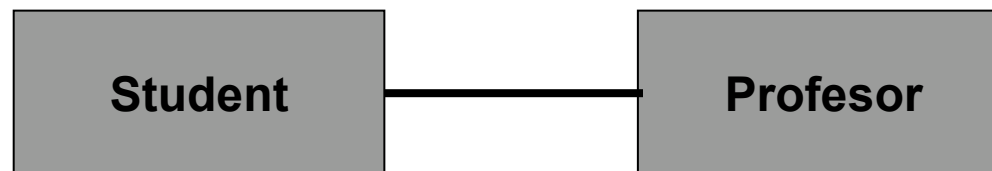
```
class Persoana{ ...  
    };  
class Student :  
    public Persoana{  
    ... };
```



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.

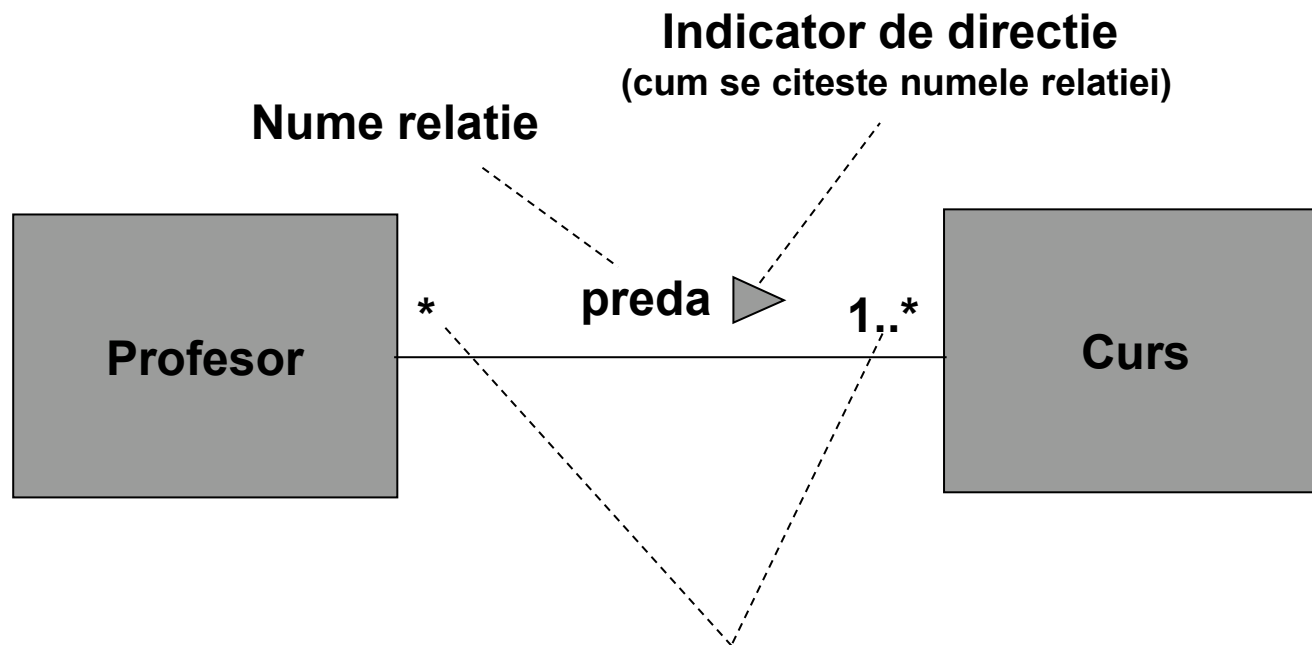
Reprezentare: — sau →



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 attribute a caror valoare serveste pt partajarea ansamblului de obj. participante la o asociere

Asocierea - exemplu



Multiplicitate (defineste numărul de obiecte asociate cu o instanță a asocierii)

Multiplicitatea

O clasa poate fi legata de alta printr'o relatie:

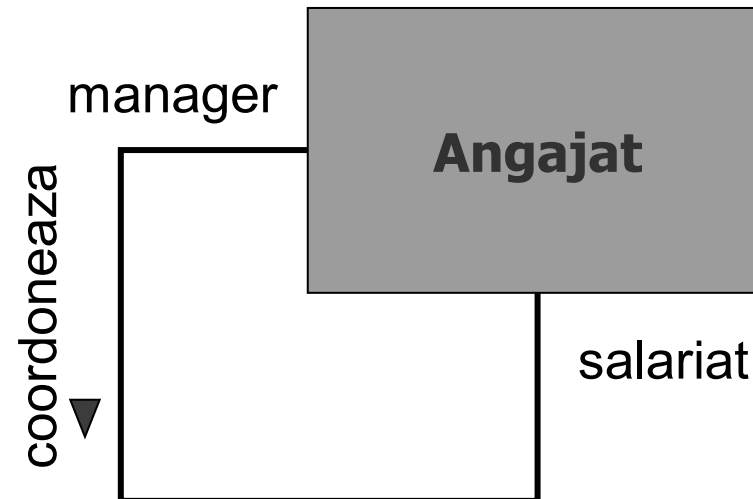
- One-to-one
- One-to-many
- One-to-one or more
- One-to-zero or one
- One-to-a bounded interval (one-to-two through twenty)
- One-to-exactly n
- One-to-a set of choices (one-to-five or eight)

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

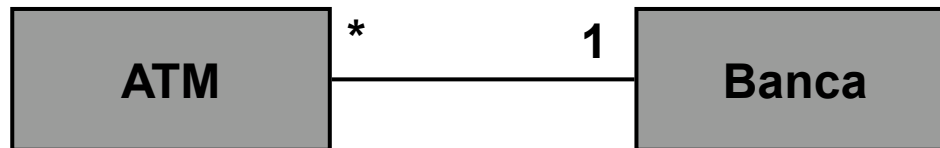
Asocierea reflexiva

- 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.

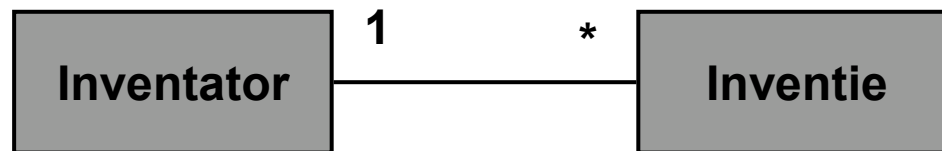


Multiplicitatea asocierilor (1)

Multi-la-unu: o banca are mai multe ATM-uri; 1 ATM este legat doar la o banca

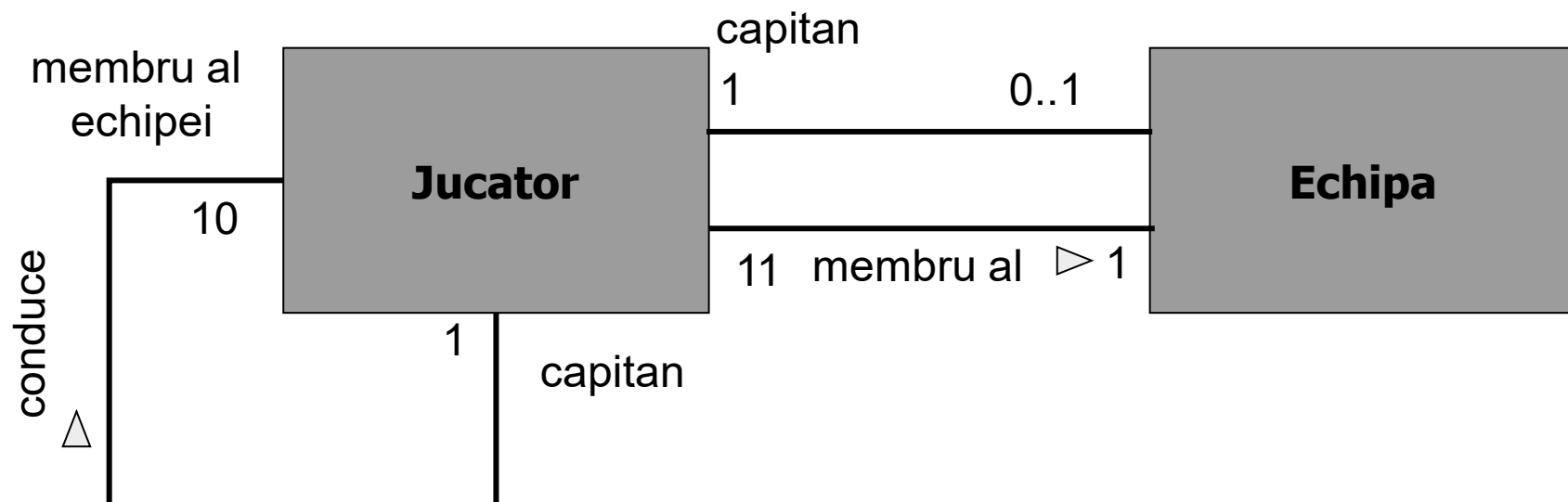


Unu-la-multi: un inventator are mai multe inventii, o inventie este a unui singur inventator

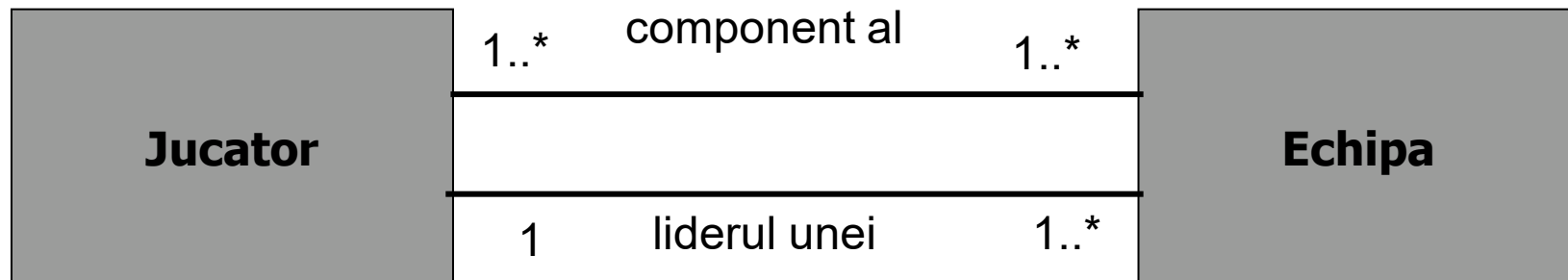


Multiplicitatea asocierilor (2)

- 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.

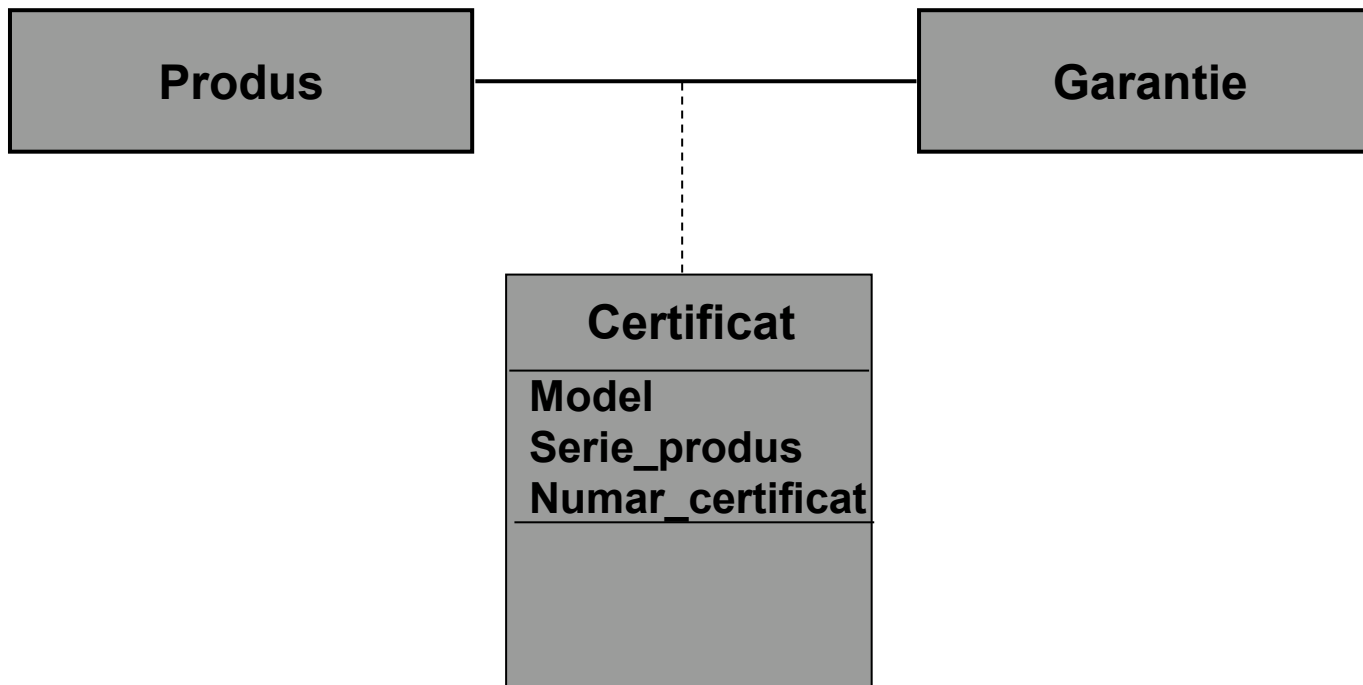


Asocieri duale



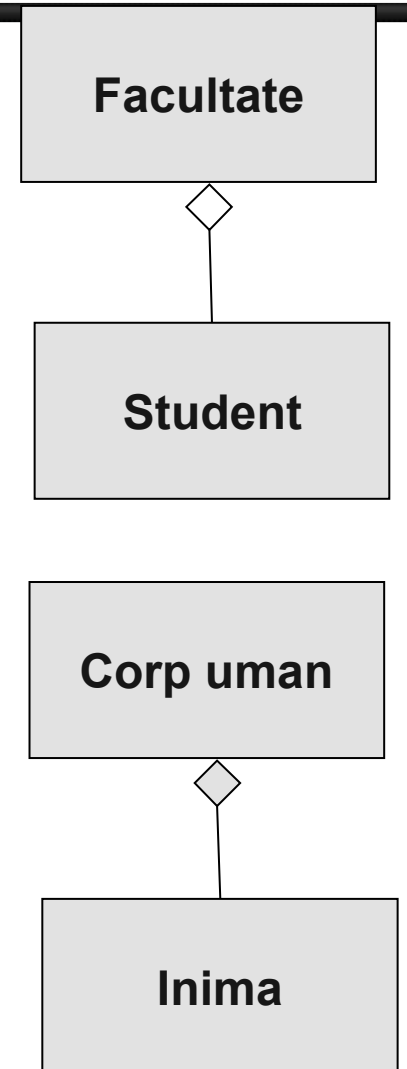
Clasa unei asocieri

- Asocierile pot fi si obiecte in sine si reprezinta instantieri ale unor clase (*link classes / association classes*).



Asociere – agregare - compunere

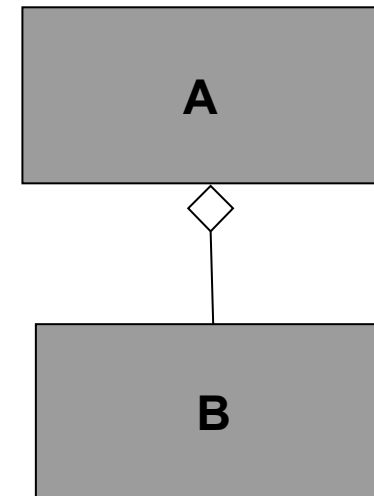
- **Agregarea** defineste o relatie "parte–intreg", in care "partea" poate exista in afara "intregului".
- Este introdusa o relatie de tip: "**has a**".
- **Compunerea** este un tip special de agregare: intregul contine mai multe parti, iar partile nu pot exista in afara intregului. Elementele componente "traiesc" si "mor" odata cu intregul



Agregare – reprezentare in C++

```
class B { ... };  
class A  
{  
private:  
  A * a;  
};
```

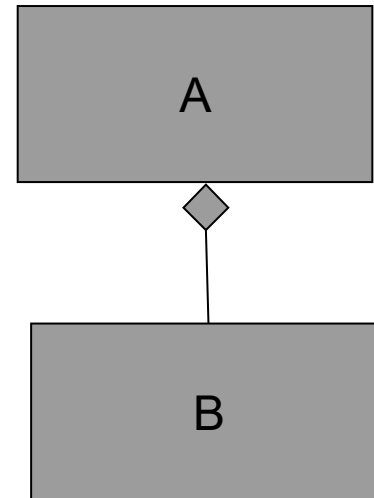
- Observatie: **a** este instantierea lui **A**



Compozitie – reprezentare in C++

```
class B { ... };  
class A  
{  
private:  
  A a;  
};
```

- Observatie: **a** este instantierea lui **A**



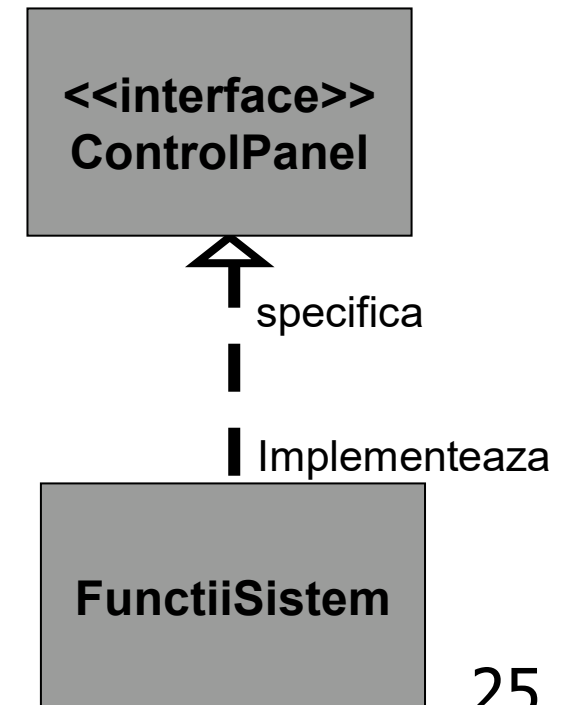
Realizare

- O relatie semantica intre doua elemente, in care unul dintre elemente "garanteaza" finalizarea actiunilor asteptate din partea celuilalt element.

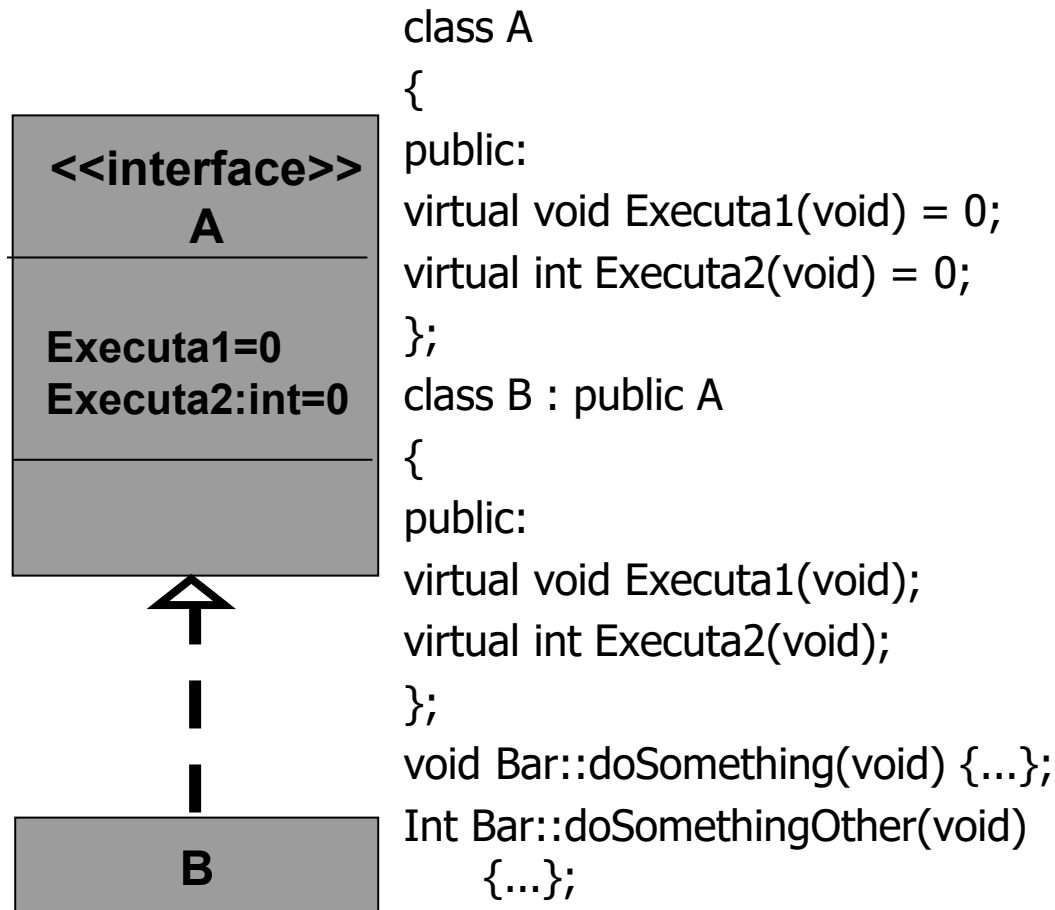
Realizare . - - - ➤

- **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



Interfata – implementare in C++



Nu este realizabila in aceasta forma.

Se poate implementa o clasa virtuala (fara attribute, fara metode private / protejate, fara implementari de metode)

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.

Diagrama de secvente (1)

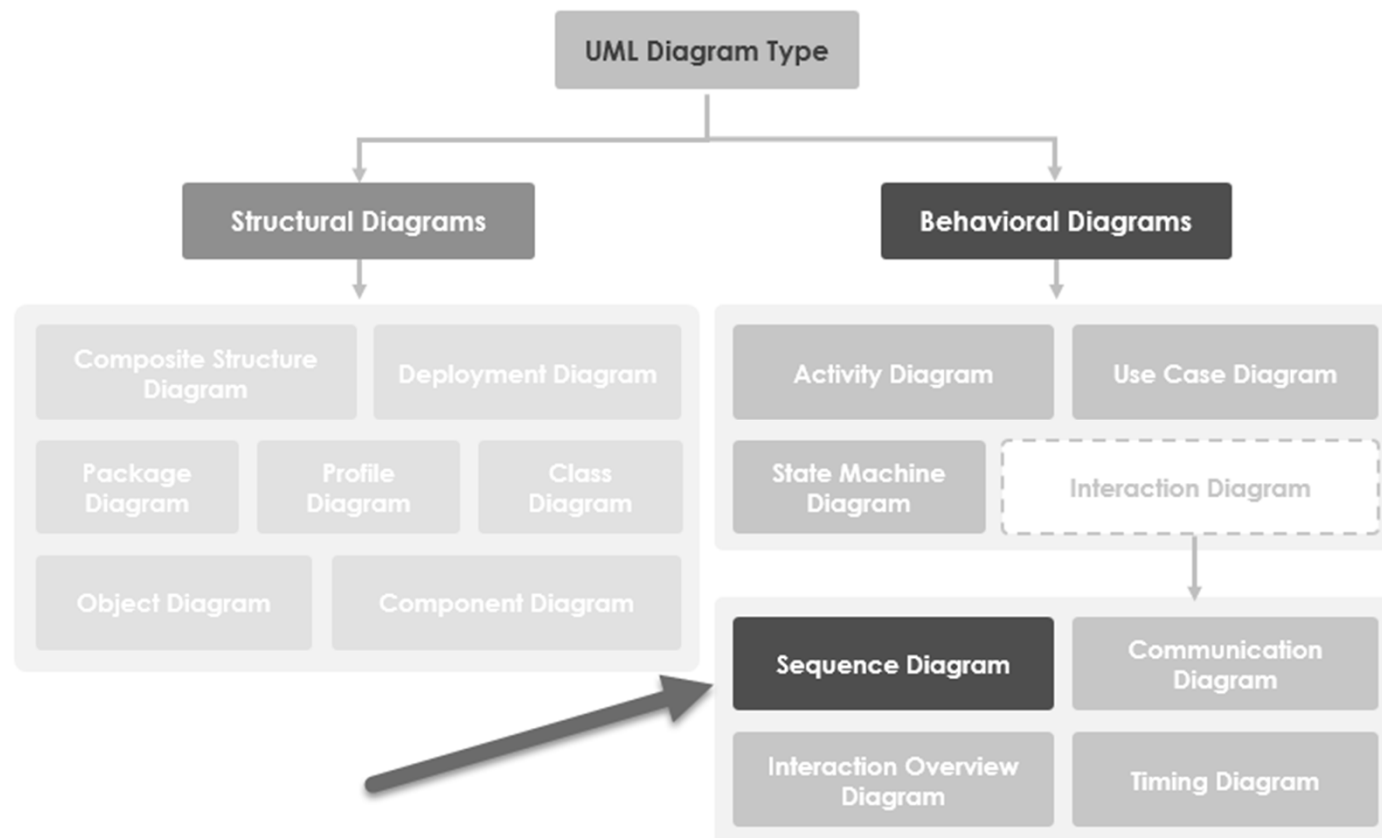


Diagrama de secvente (2)

- **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:* reprezentate prin "sageata plina"; durata trebuie sa fie indicata prin bara de activare sau sageata de intoarcere
 - *Asincrone:* reprezentata prin "jumătate de sageata"
 - Pot exista mesaje de tip «create» si «destroy»

Diagrama de secvente (3)

- **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*)



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 redat cu sageti orizontale ("coboara" odata cu trecerea timpului)
- Condițiile (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)

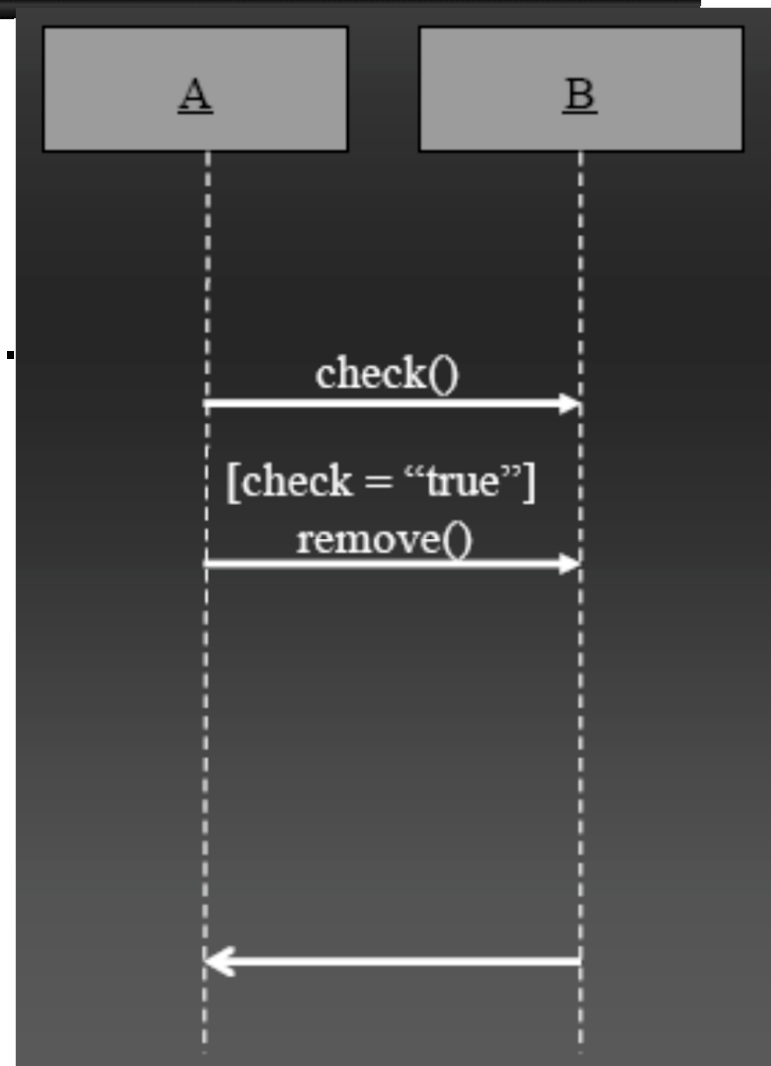


Diagrama de secvente - observatii

- **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** (***K**eep **I**t **S**mall & **S**imple*)

Diagrama de activitati (1)

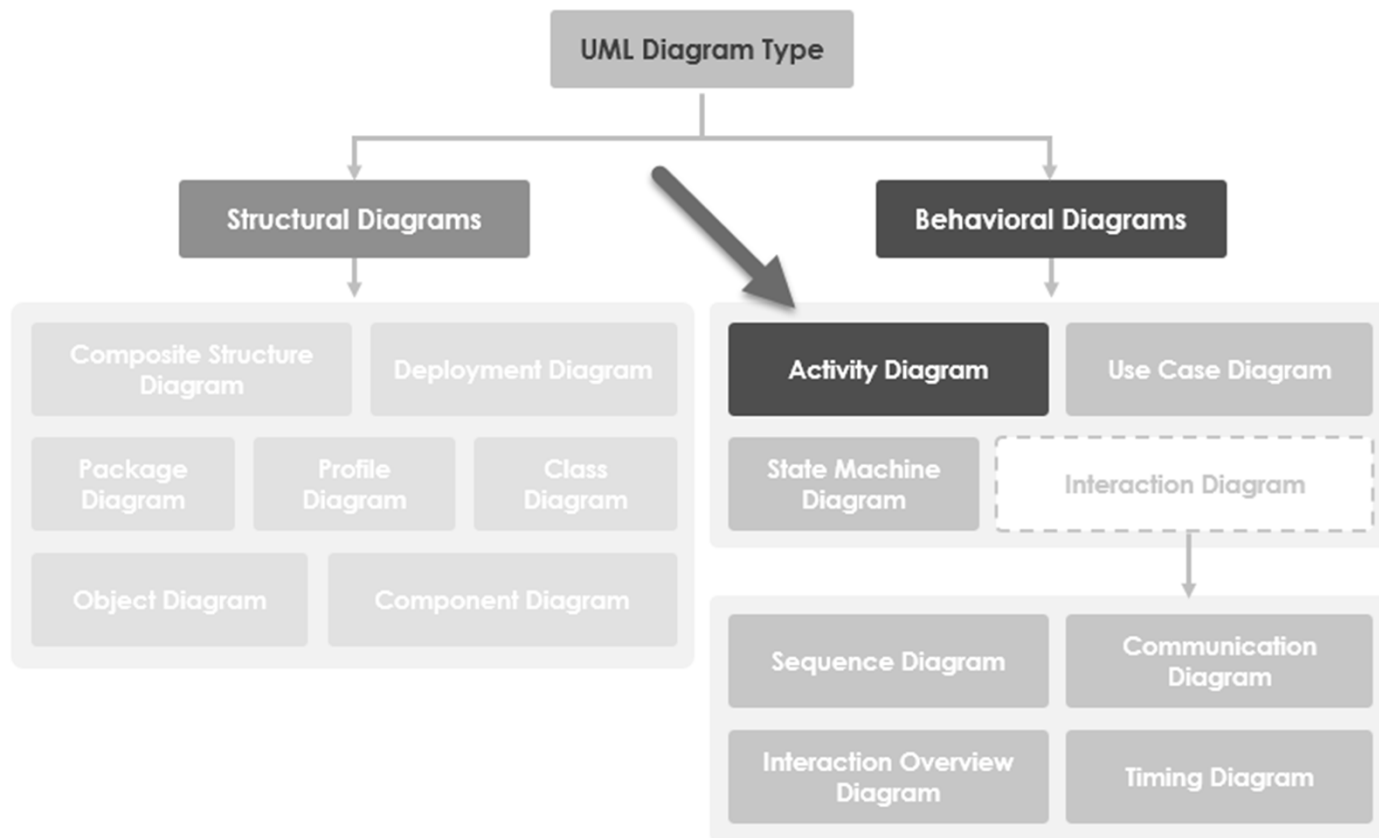


Diagrama de activitati (2)

- **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

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 sistem
- Diagrama de activitati este un caz special al diagramei de stari in care starile sunt inlocuite cu activitati (functii).

Exemplu

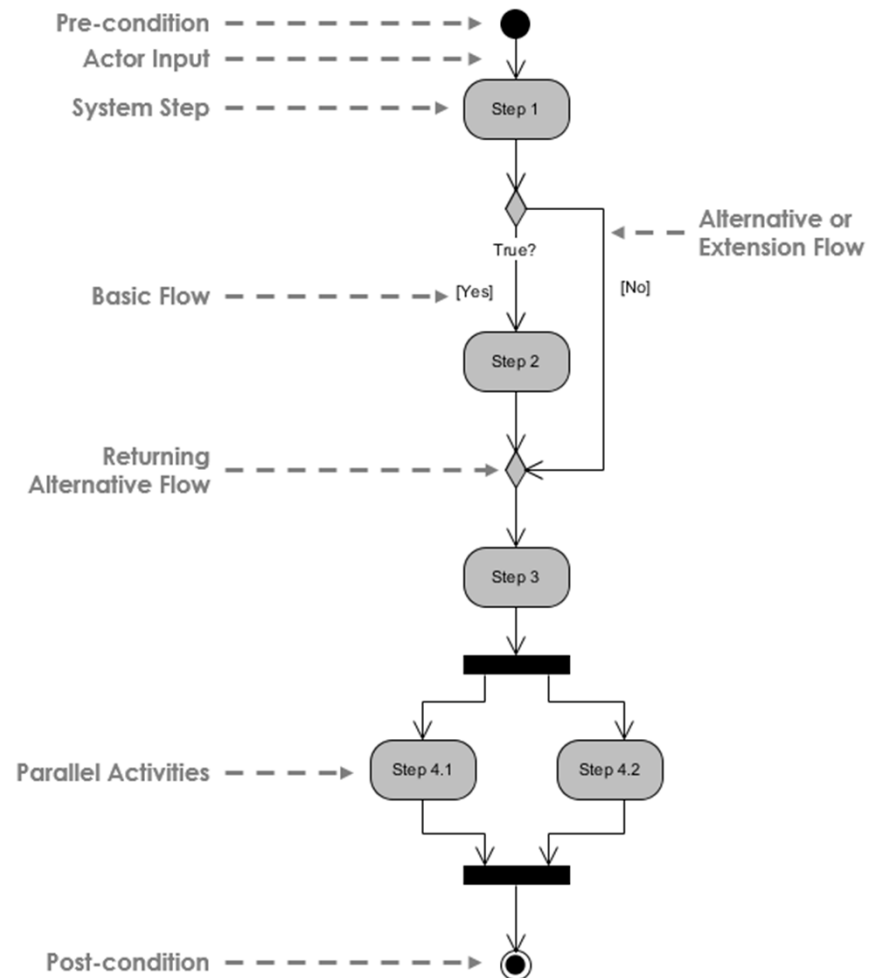


Diagrama de colaborari (1)

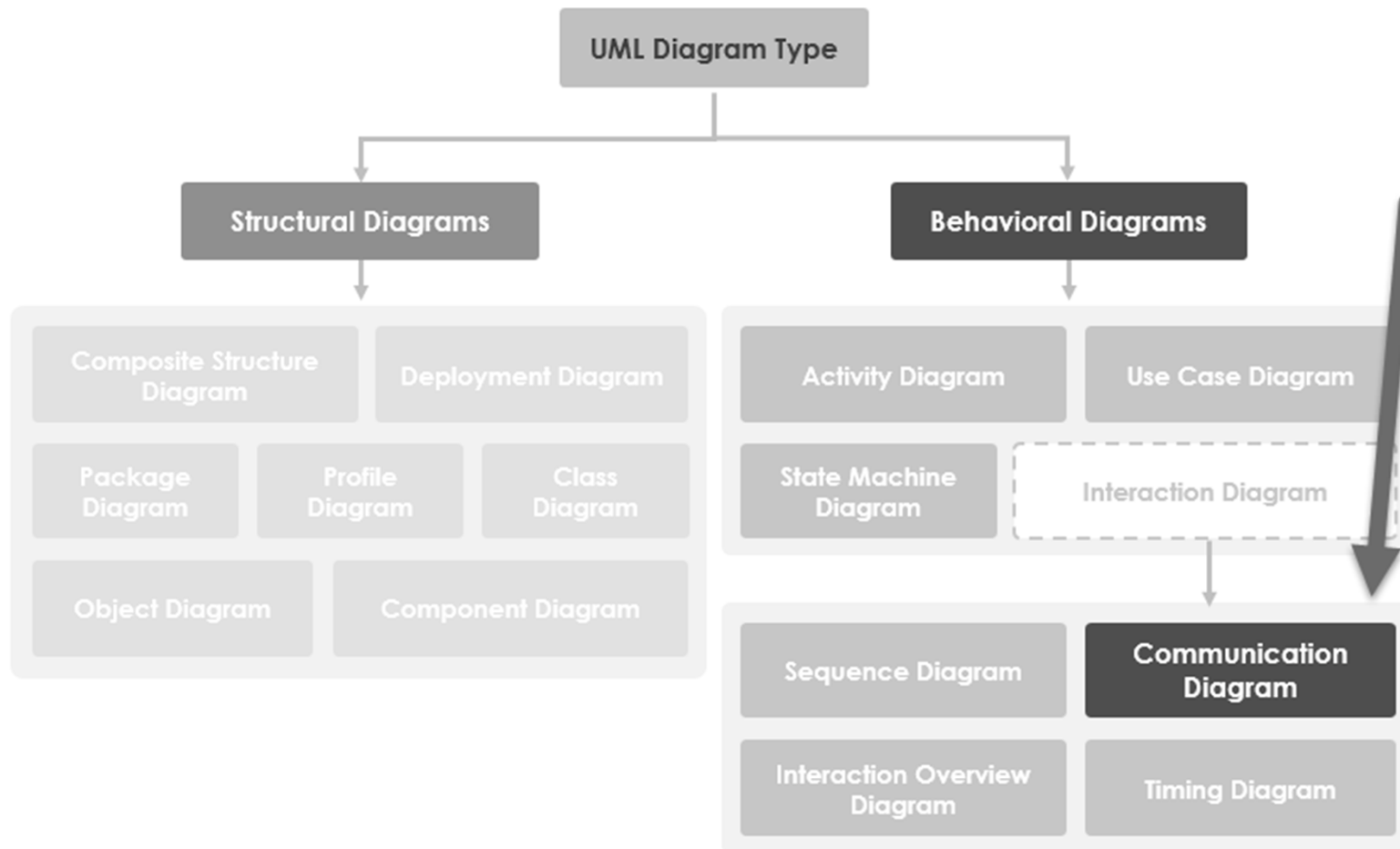


Diagrama de colaborari (2)

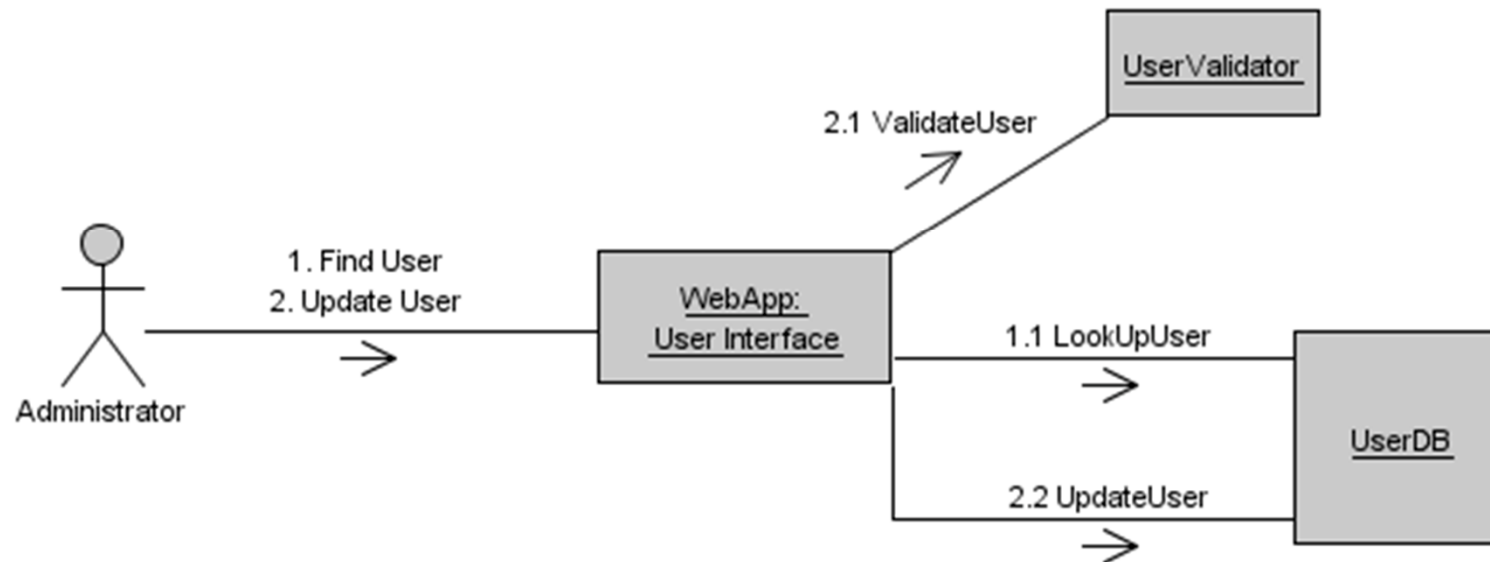
- Prezinta modul in care obiectele interactioneaza unele cu altele (tinand cont de unitatile organizationale)
- Secventa de mesaje este determinata prin numere:
 - 1, 2, 3, 4,
 - 1, 1.1, 1.2, 1.3, 2, 2.1, 2.1.1, 2.2, 3 (prezinta modul in care o operatie apeleaza o alta operatie)

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"

Exemplu

1. Find User
 - 1.1 LookUpUser
2. Update User
 - 2.1 ValidateUser
 - 2.2 UpdateUser



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
- **Observatie:** Controlul complex este greu de reprezentat printr'o singura diagrama!!!

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

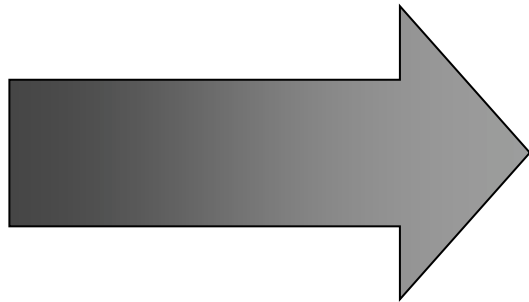
5. **Diagrama de activitati**

[Model Dinamic]

→modeleaza fluxul de control (ca tranzitie intre activitati)

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*)

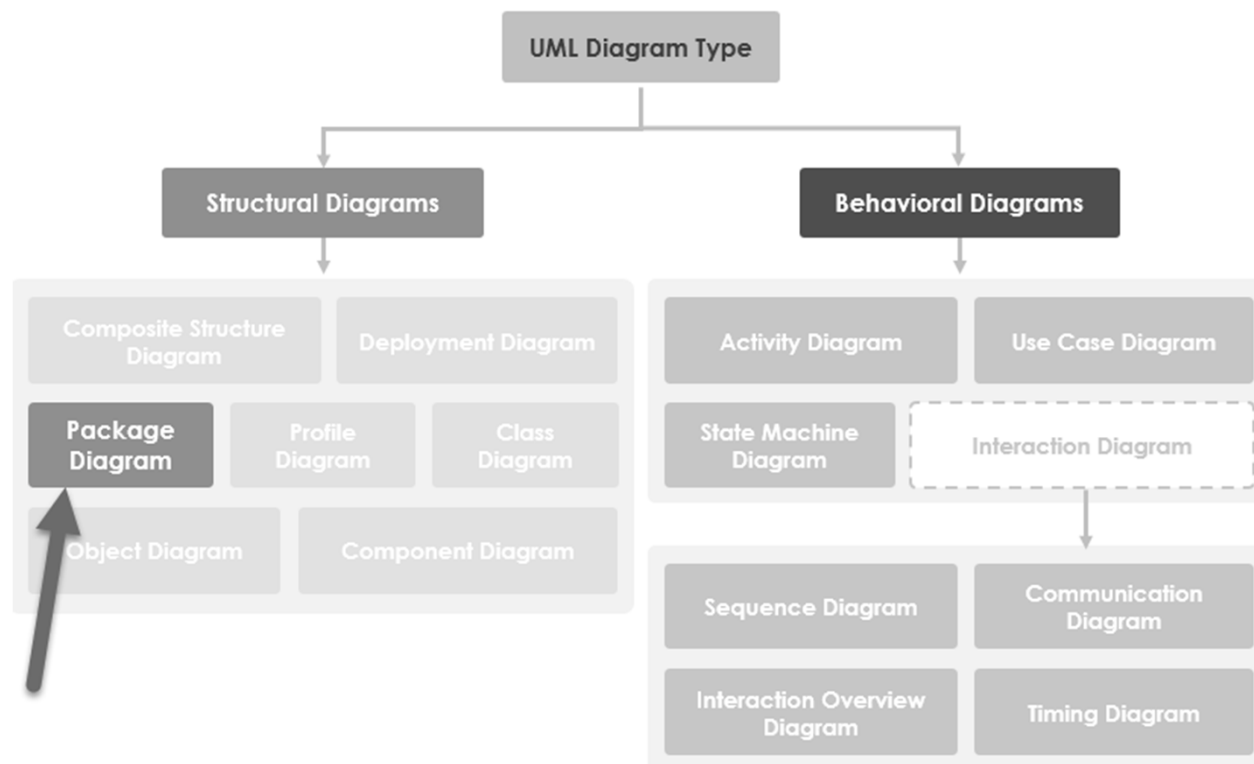
(conceptual: in momentul dezvoltarii sistemului)

Componente

(fizic: in momentul rularii)

Pachete (*Packages*)

Un **pachet** reprezinta o "grupare" de elemente de modelare;
poate contine clase sau alte pachete / diagrame



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*)

Pachete - exemple

Vanzari

Client

Ordin

Depozit

Locatie

Item

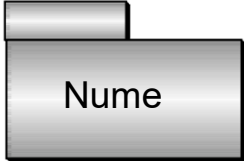


Stoc

Ordin

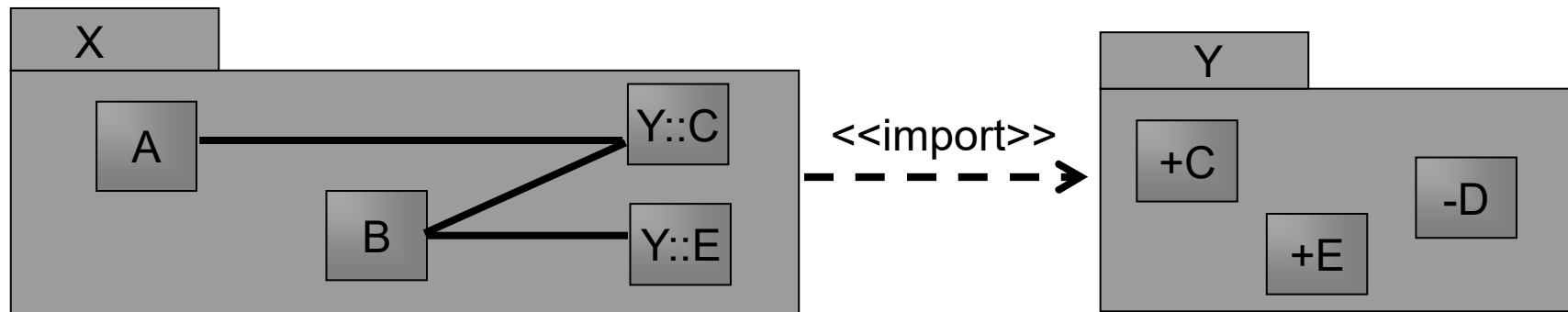
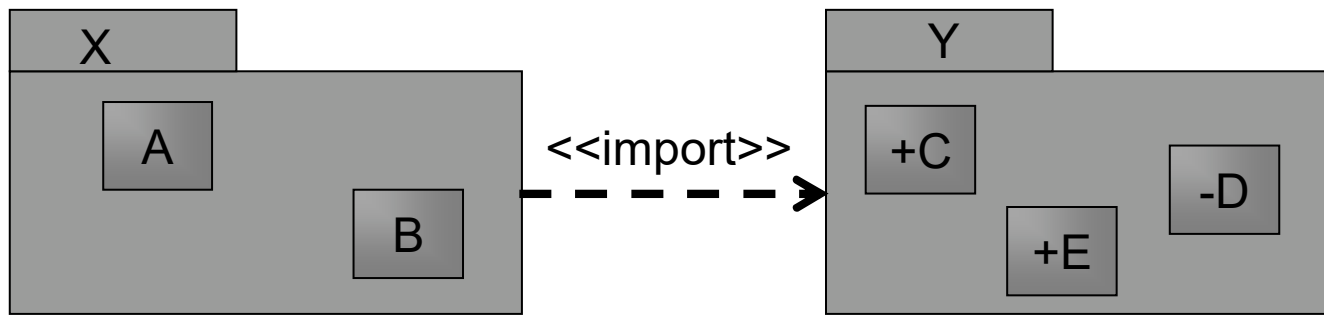
Pachete – caracterizare

- 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.

Concepte de baza

<i>Package</i>	Grup de elemente de modelare.	
<i>Import</i>	Relatie de dependenta; indica faptul ca elementele de continut ale pachetului tinta sunt adaugate spatiului pachetului sursa (extinde spatial de lucru al P importator)	<<import>> 
<i>Access</i>	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).	<<access>> 

Relatii intre pachete



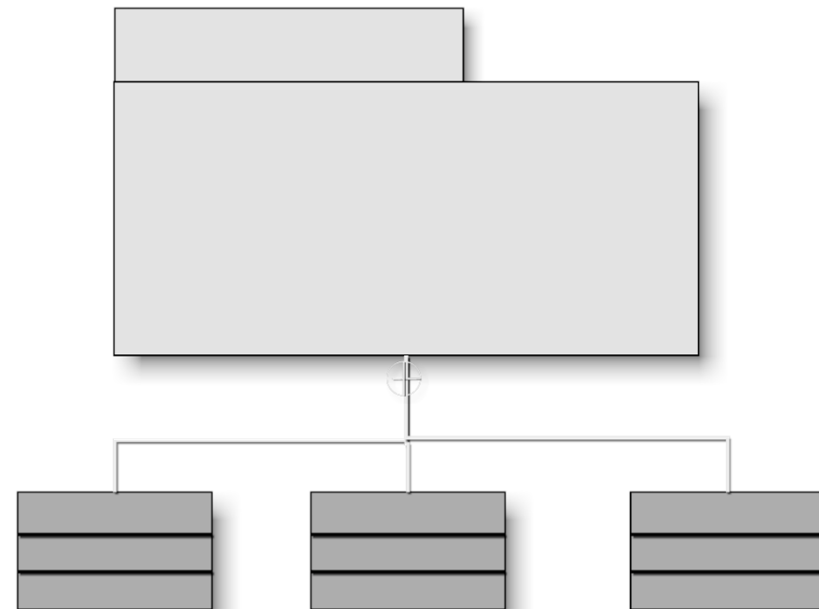
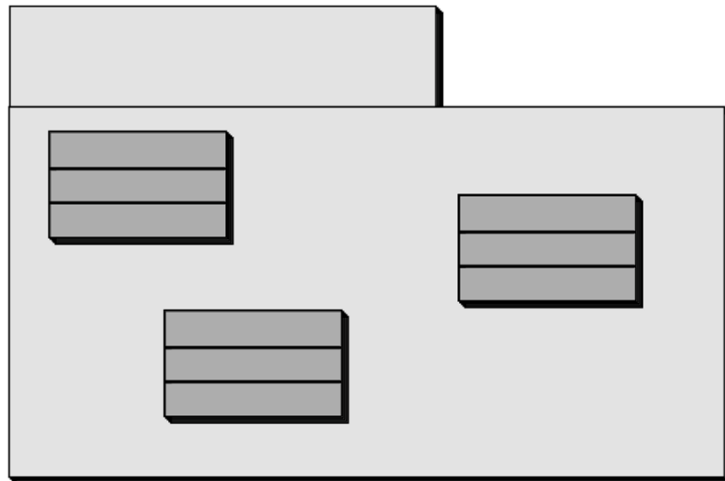
Mostenirea

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

Reprezentarea pachetelor (continut)

- Pachetele sunt reprezentate prin intermediul diagramelor statice
- Exista doua modalitati echivalente de reprezentare:



Utilizarea pachetelor

- Cerinte
 - ➔ Grd cat mai inalt de coeziune interna
 - ➔ Nivel cat mai redus de cuplare externa
 - ➔ Unitatea scopului

Utilizari

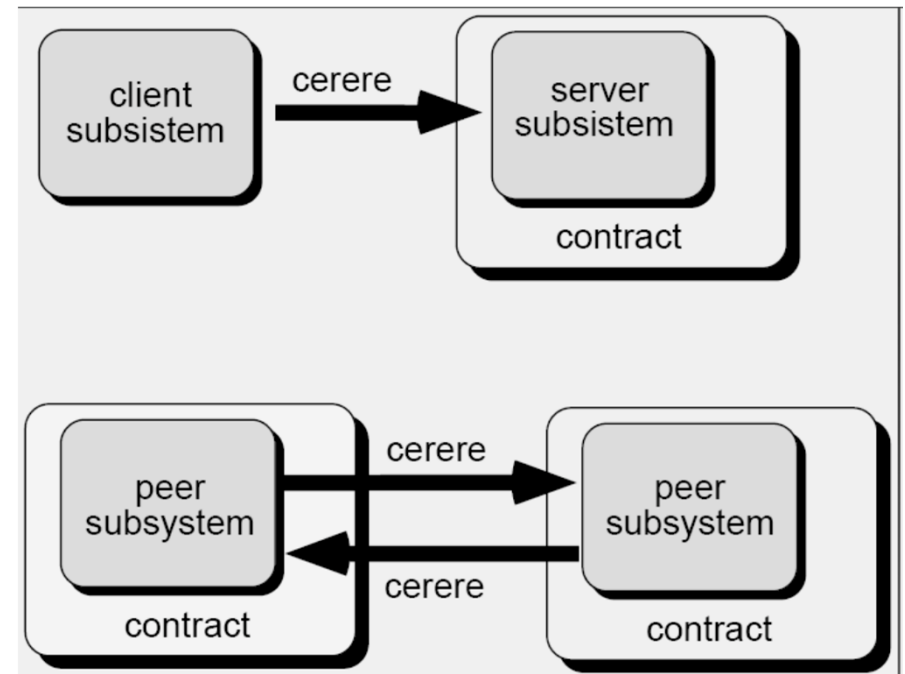
- Crearea unei “vederi de ansamblu” asupra unui mare numar de elemente de modelare
- Organizarea modelelor de mari dimensiuni
- Gruparea elementelor interconectate
- Separarea spatiilor de lucru

Pachete – reguli de configurare

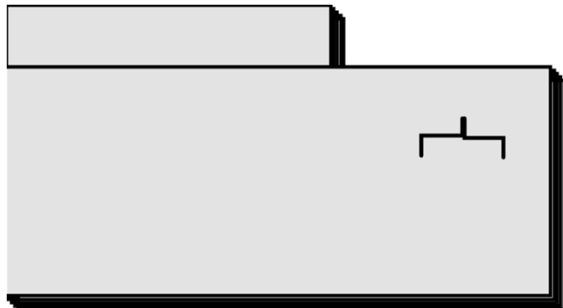
- Elementele de modelare puternic cuplate trebuie sa se gaseasca in acelasi pachet
- Elementele de modelare slab cuplate trebuie sa se gaseasca in pachete diferite.
- Trebuisc evitate relatiile (in special asocierile) dintre elementele de modelare aflate in pachete diferite.
- Un element importat intr'un pachet nu trebuie sa "cunoasca" cum este folosit in acel pachet.

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

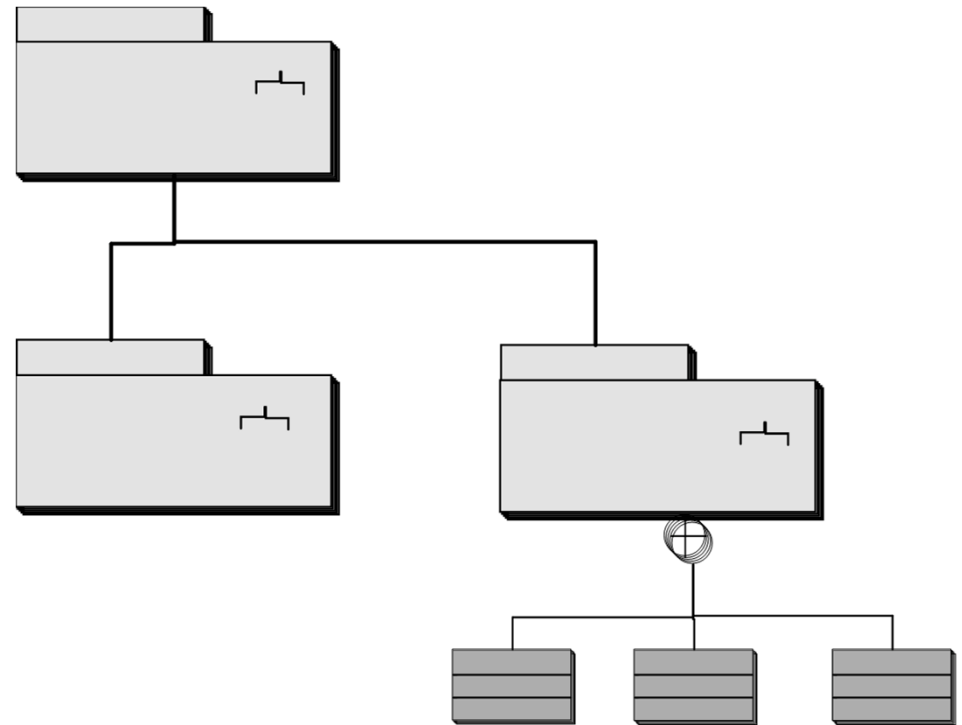


Subsisteme

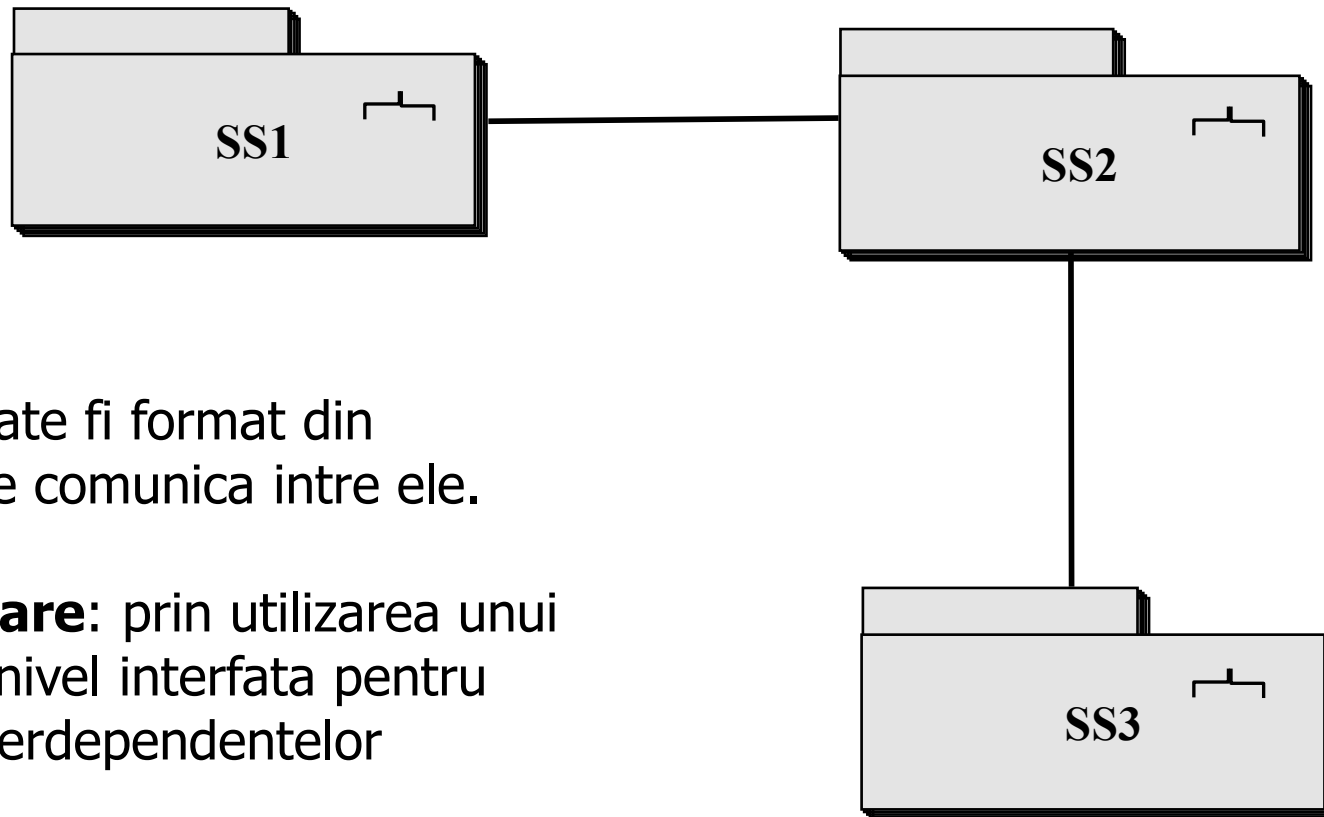


Subsistemele sunt utilizate in descompunerea sistemelor.

Grup de elemente de modelare ce formeaza impreuna o unitate de comportament in sistem fizic



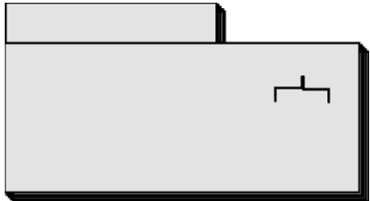
Subsistem – exemplu



Un sistem poate fi format din subsisteme ce comunica între ele.

Interconectare: prin utilizarea unui subsistem la nivel interfata pentru reducerea interdependentelor

Concepte de baza

Subsystem	Un grup de elemente de modelare ce reprezinta o unitate de evolutie intr'un sistem fizic.	
------------------	---	---

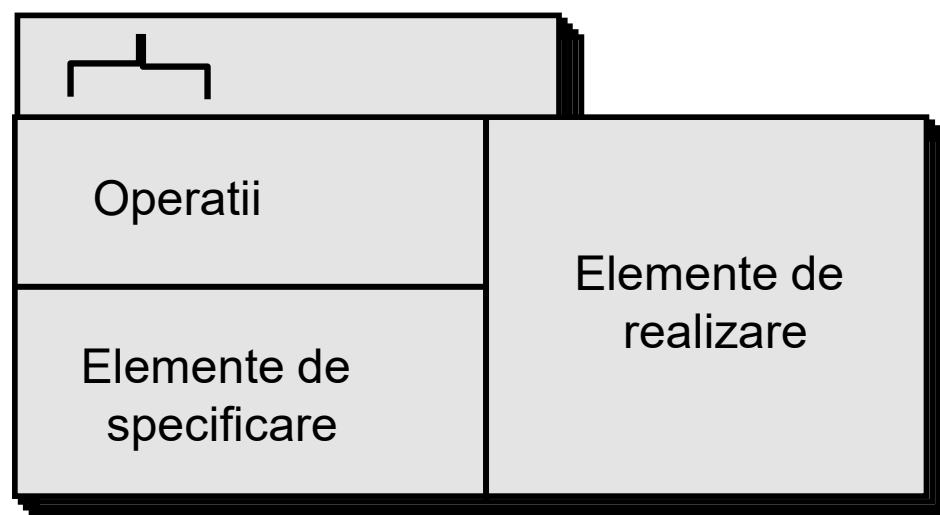
Subsystemele se definesc dpdv: *functional*, *logic* si *avand coeziune fizica*

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.

Caracteristicile subsistemelor

Un subsistem are elemente de specificare, respectiv de realizare pentru a putea implementa cele doua componente ale sale

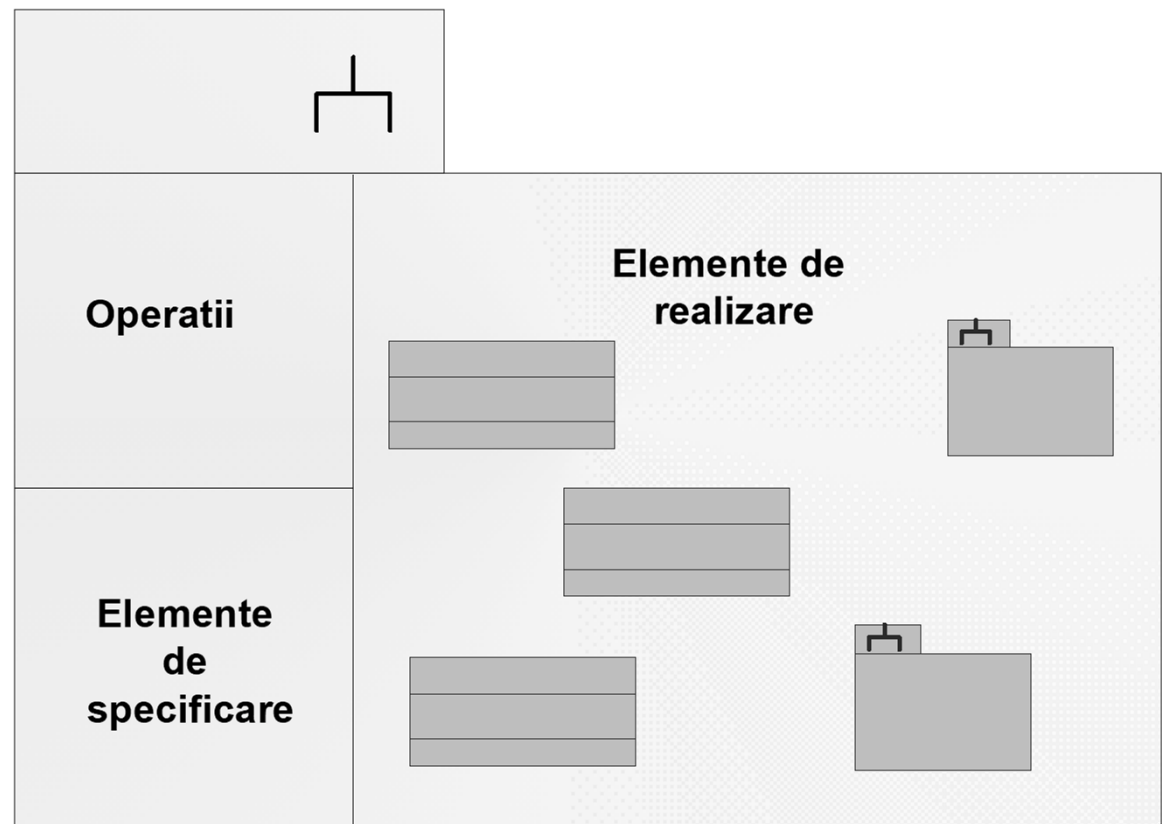


Elemente de realizare

→ Elementele de realizare definesc continutul actual al subsistemului.

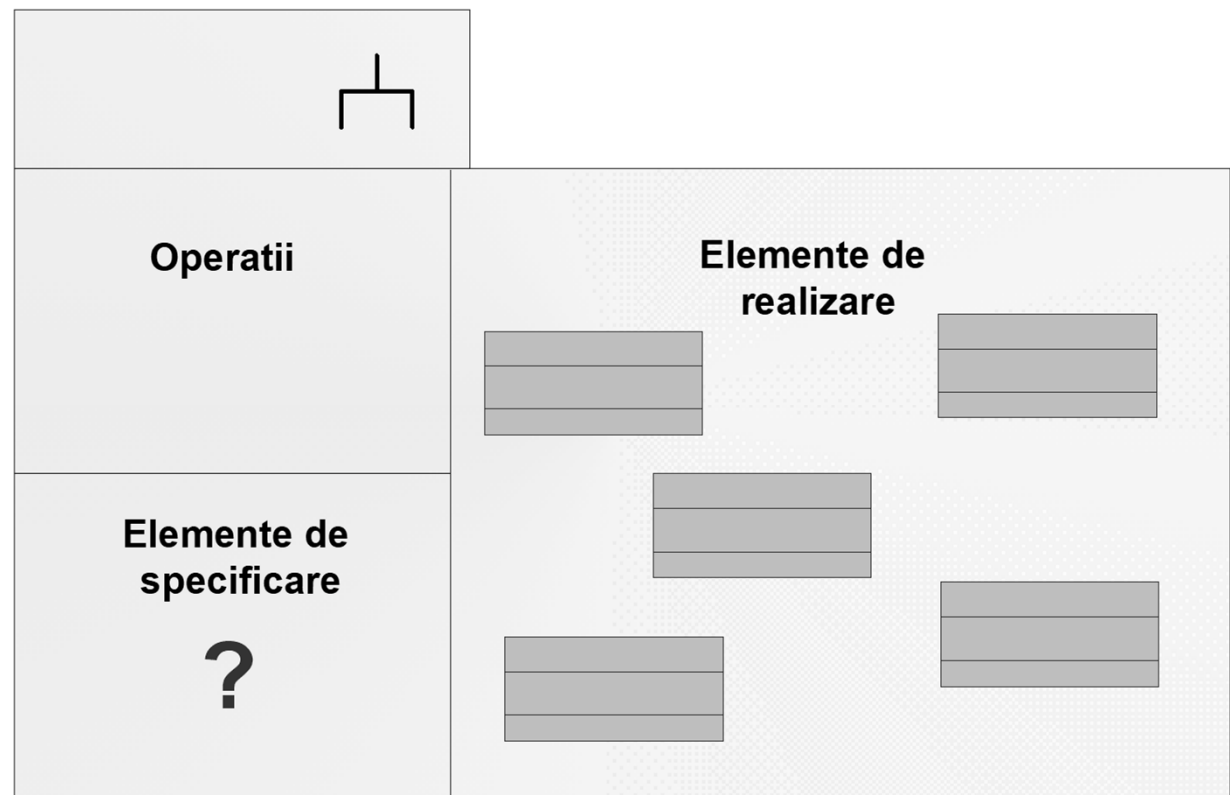
→ Elementele de realizare consta in:

- Clase si relatiile dintre ele
- Ierarhii de subsisteme



Elemente de specificare (1)

- Elementele de specificare definesc vizibilitatea externa a subsistemului



Elemente de specificare (2)

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

Cand se utilizeaza subsistemele ?

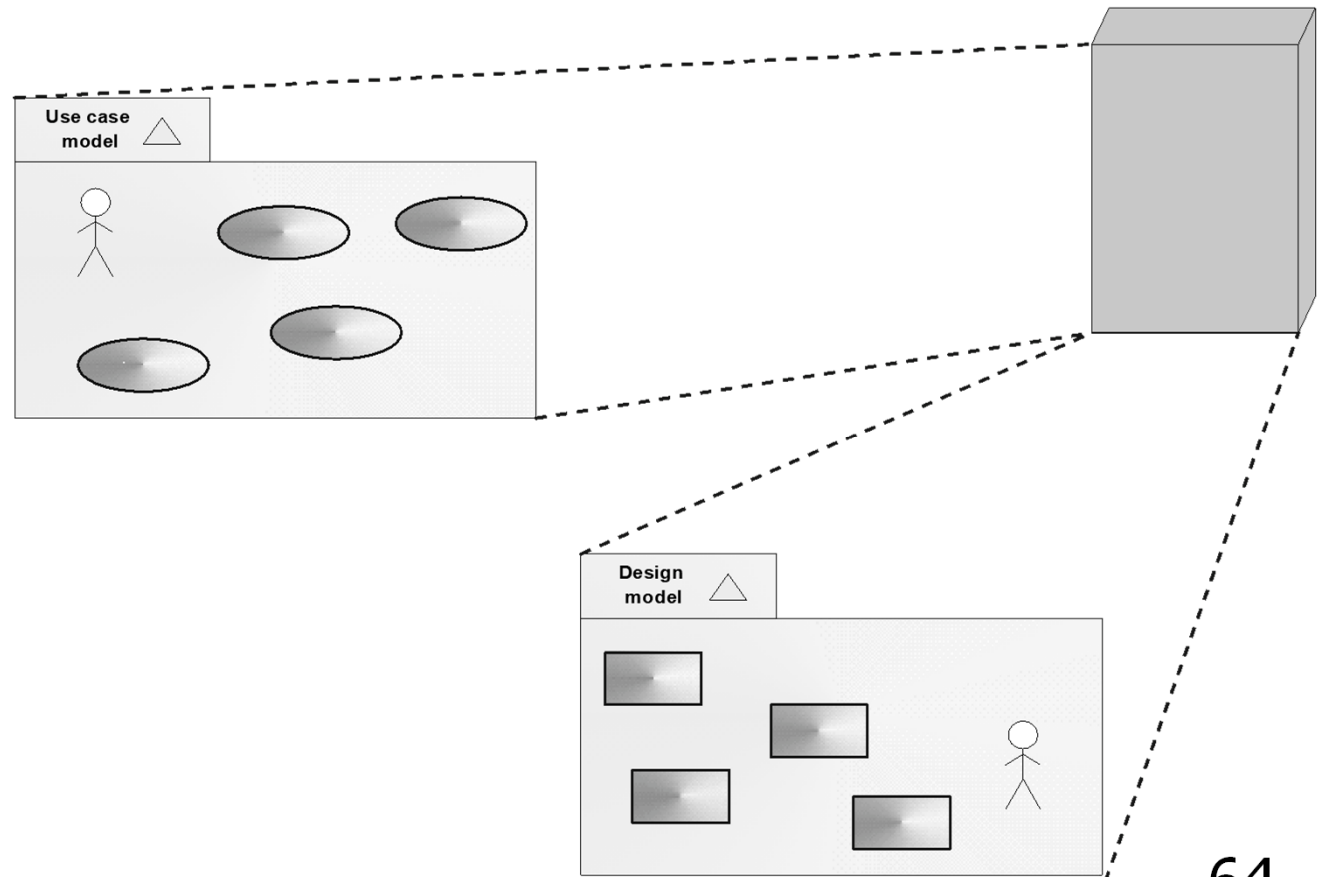
- Descompunerea sistemelor mari in module / componente mai mici
- Dezvoltarea sistemelor distribuite
- Asamblarea unui set de module intr'un sistem de dimensiuni mai mari
- Dezvoltarea sistemelor bazate pe structuri multiple

Subsisteme – reguli de configurare

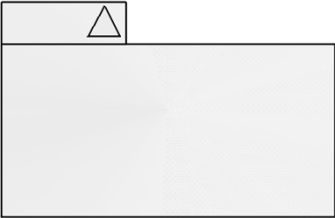
- Un subsistem se definește pentru fiecare parte a unui sistem de mari dimensiuni
- Specificațiile tehnice sunt dependente de tipul sistemului / subsistemului
- Fiecare subsistem trebuie să fie proiectat în mod independent

Modele

- Un model surprinde o "vedere" asupra unui sistem si incearca o "descriere" completa a acestuia



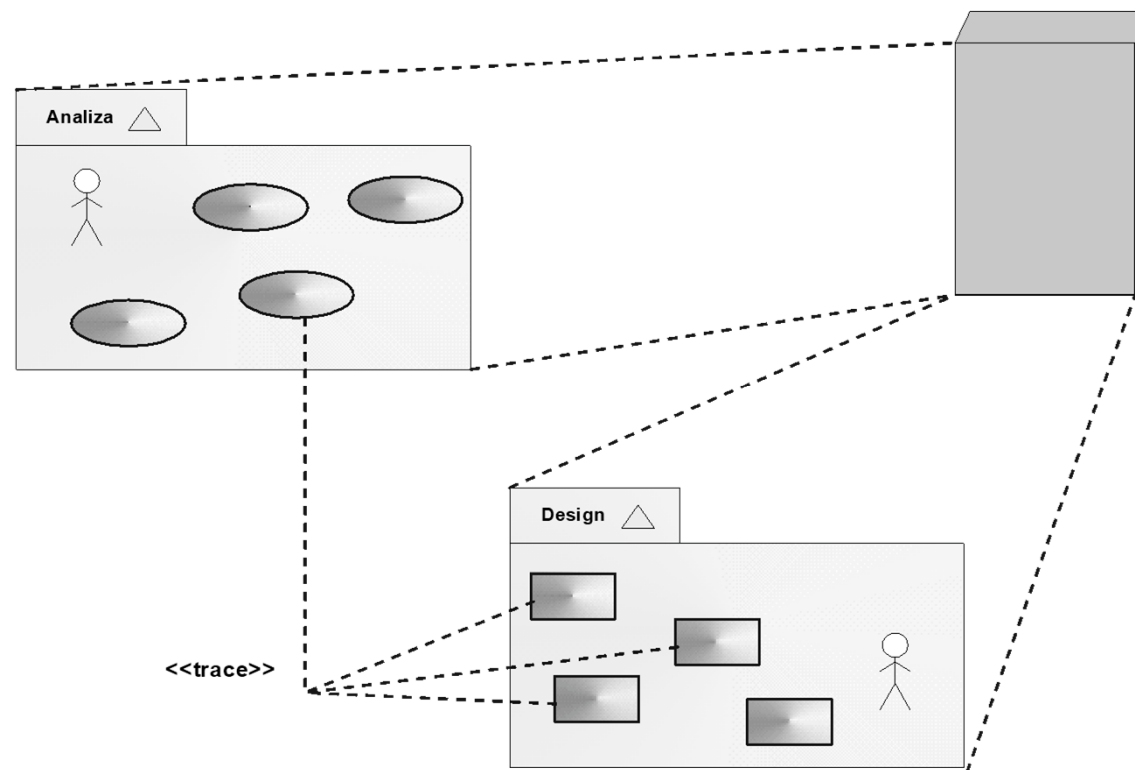
Concepte de baza

Model	O viziune asupra unui sistem, intr'un anumit scop, in vederea determinarii aspectelor semnificative ale sistemului	
<i>Trace</i>	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)	<p><<trace>></p> <p>-----</p>

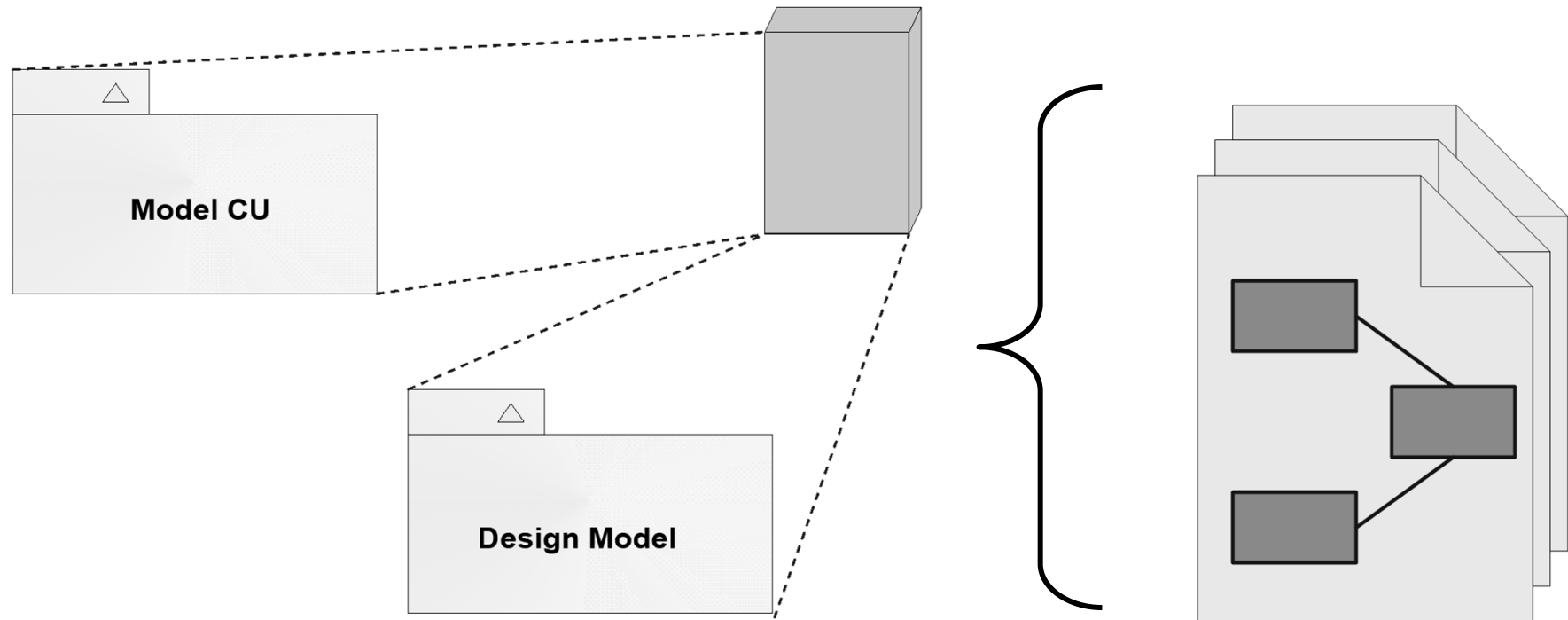
Trace

Este un stereotip
al relatiei de
dependenta

Nu se aplica
elementelor ce
apartin aceluiasi
model



Modele *vs.* diagrame



- Diagramele specifica detaliile tehnice ale unui model.

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