
Analiza și proiectarea sistemelor software

Curs 5

Model

- Model – descriere a unui sistem abstractizând ceea ce este relevant dintr-o anumită perspectivă și cu un anumit scop.
- Construire model - descompunere, abstractizare și ierarhie.
- Evaluare model :
 - În condiții tipice
 - În condiții neașteptate
- Modificare corespunzător rezultatelor evaluării

Metodă și metodologie

- Metodă – *procedură disciplinată* pentru generarea unui *set de modele* ce descriu *diferite aspecte* ale unui sistem software în curs de dezvoltare folosind o *notație* bine definită.
- Metodologie – *colecție de metode* aplicate de-a lungul ciclului de dezvoltare de software și *unificate* prin *proces, practici* și o *abordare* (filozofie) generală.

Metodologii pentru proiectare software

- în continuă evoluție, în curs de maturizare

Elemente comune:

- **Notăție:** limbajul pentru exprimarea fiecărui model
- **Proces:** activitățile ce conduc la construirea ordonată a modelelor sistemului
- **Instrumente:** artefactele utilizate în construirea modelului (maschează rutina, impun respectarea de reguli de modelare, detectează erori și inconsistențe).

Procesul de proiectare nu este deterministic: diferiți proiectanți pot produce *modele diferite* pentru *soluția aceleiași probleme*.

O metodologie completă de proiectare se bazează pe un *fundament teoretic solid*, dar oferă *grade de libertate pentru inovație*.

REPREZENTARE MODELE

MOTTO:

If you follow the work of any engineer you will soon realize that the one and only place that a system is conceived is in the mind of the designer. As this design unfolds over time, it is often captured on such high-tech media as whiteboards, napkins, and the backs of envelopes.

- Definirea structurii conceptuale – activitate esențial umană.
- Exprimarea acesteia – utilizând notații.

Notația – capturează modelul (analiză sau proiect) dându-i o reprezentare ce poate fi comunicată.

Utilitatea unei notații bune, standard:

- comunicare *neambiguă* a deciziilor de proiectare
- eliberează mintea de probleme de *rutină* permițând concentrarea pe probleme avansate
- elimină din efortul de *verificare* a consistenței și corectitudinii modelelor

REPREZENTARE MODELE

INSTRUMENTE pentru reprezentare modele.

Great designs come from **great designers**, not from great tools. Tools simply empower the individual, freeing him or her to concentrate on the truly **creative aspects** of analysis or design. Thus, there are some things that tools can do well and some things that tools cannot do at all.

Facilități tipice:

- editare modele
- verificare consistență modele
- verificare constrângeri
- verificare completitudine
- navigare liberă printre produsele analizei și proiectării

UML - Unified Modeling Language

UML - limbaj de modelare utilizat pentru *analizarea, specificarea și proiectarea* sistemelor software.

- Bazat pe orientarea obiect, dar cu facilități suplimentare pentru fațete utile preluate din alte metodologii.
- Utilizat pentru *reprezentarea modelului sistemului* ce urmează a fi construit.
- Modelul reprezintă, cu un anumit *grad de fidelitate*, sistemul.
- Modelul este compus din mai multe *diagrame*, fiecare oferind o anumită *perspectivă* asupra sistemului.

UML - Unified Modeling Language

Din specificația 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 - istoric

Booch, Rumbaugh, și Jacobson

- mijlocul anilor '90
- Rational Software Corporation
- combinarea tehnologiilor pe care le utilizau

Colaborare cu alți specialiști și companii



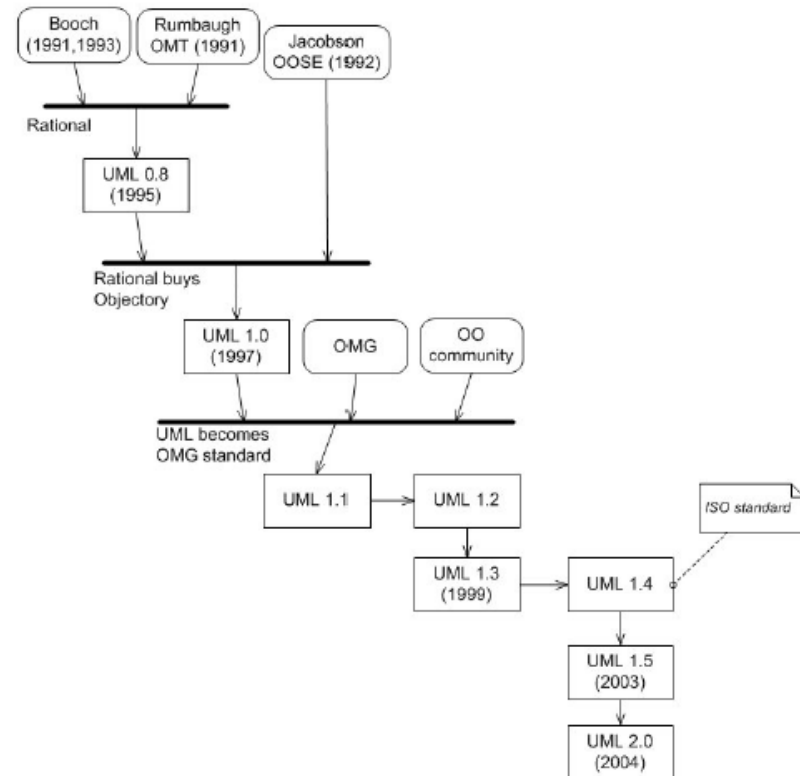
Propunere limbaj standard de modelare, la OMG

- noiembrie 1997 – adoptat ca standard,
- OMG a preluat gestionarea și dezvoltarea acestui standard.

OMG – consorțiu ce crează și gestionează standarde pentru industria calculatoarelor.

<https://www.omg.org/spec/UML/>

<https://www.omg.org/spec/UML/2.5.1/PDF> - 2017



UML – consistență model

Un model conține *perspective diferite* asupra sistemului ce va fi realizat, reprezentate în:

- mai multe tipuri de diagrame
- mai multe diagrame de același tip.

Consistență model = Toate entitățile de prim ordin de același tip și cu același nume din toate diagramele referă același element al modelului.

UML – categorii de entități modelate

Clasificator

entitate individuală, caracterizată de stare și relații cu alte entități.

A Classifier has a set of Features, some of which are Properties called the attributes of the Classifier. Each of the Features is a member of the Classifier.

The values that are classified by a Classifier are called instances of the Classifier.

Eveniment

apariție posibilă, având consecințe asupra sistemului.

An Event is the specification of some occurrence that may potentially trigger effects by an object.

Comportament

execuție posibilă, realizată cu un set de acțiuni care pot genera sau răspunde la evenimente, incluzând accesarea și modificarea stării obiectelor.

Descrise cu construcții de modelare ale limbajului UML.

UML – categorii de entități modelate

Comportament - execuție posibilă, realizată cu un set de acțiuni care pot genera sau răspunde la evenimente, incluzând accesarea și modificarea stării obiectelor.

UML provides Behavior, Event, and Trigger constructs to model the corresponding fundamental concepts of behavioral modeling.

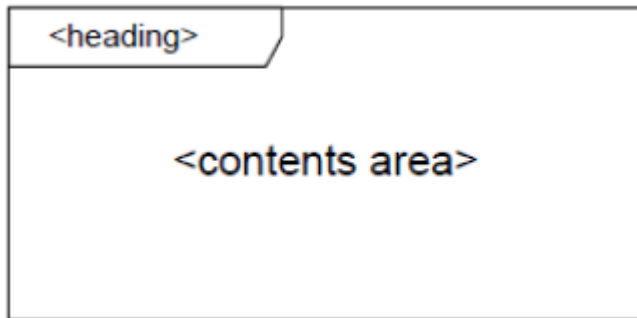
Behavior is the basic concept for modeling dynamic change. Behavior may be executed, either by direct invocation or through the creation of an active object that hosts the behavior. Behavior may also be emergent, resulting from the interaction of one or more participant objects that are themselves carrying out their own individual behaviors.

Dynamic behavior results in events of interest that occur at specific points in time. Such events may be implicit, occurring on the change of some value or the passage of some interval of time. They may also be explicit, occurring when an operation is called or an asynchronous signal is received.

The occurrence of an event may then trigger new behavior, or change the course of already executing behavior. Explicit events thus provide the basic mechanism for communication between behaviors, in which an action carried out in one behavior, such as calling an operation or sending a signal, can trigger a response in another behavior.

UML – diagrame

Diagramă UML – reprezentare grafică a unei părți a modelului UML al unui sistem software.



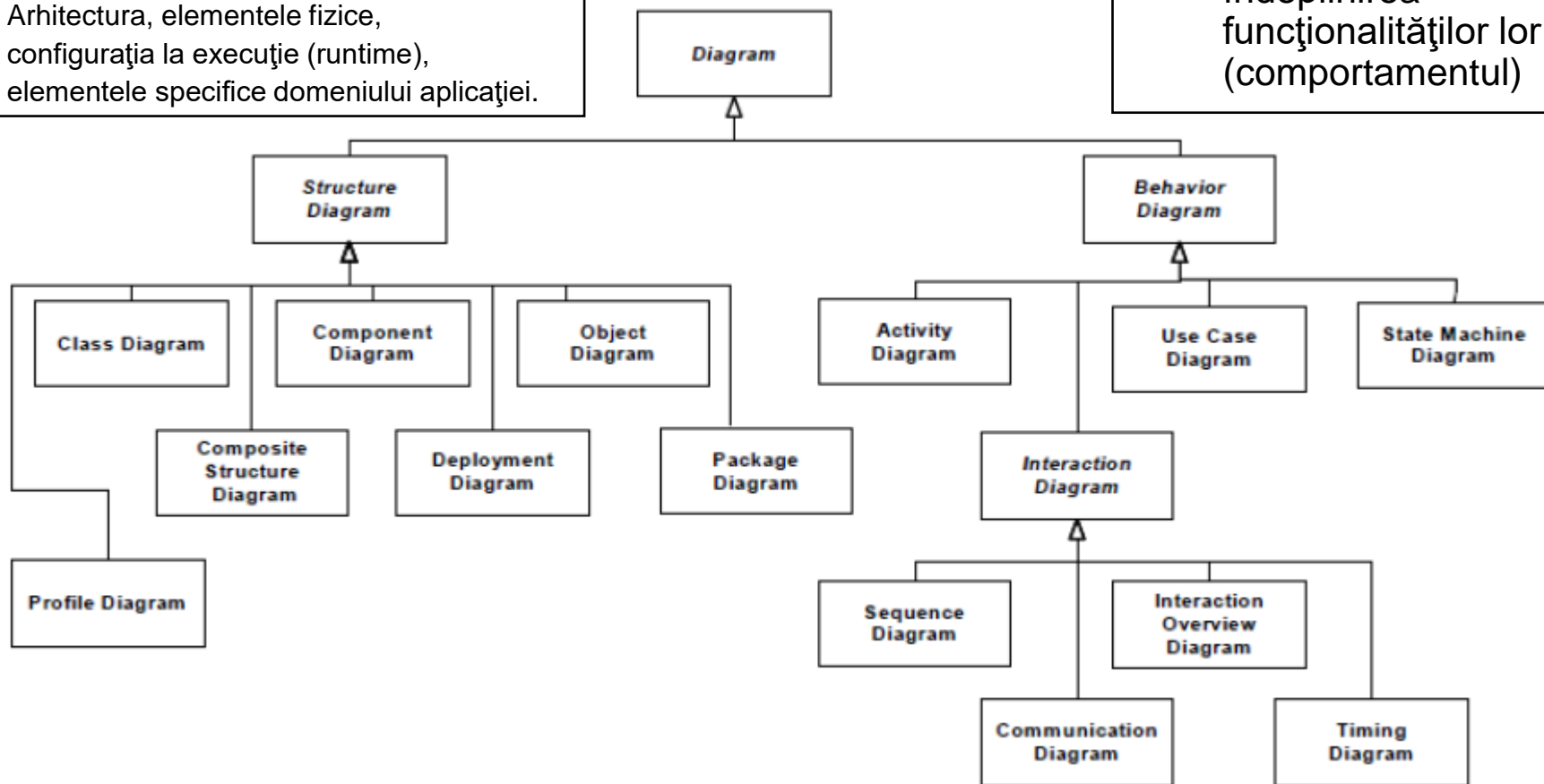
Reprezintă structura statică,
indiferentă de timp, a
elementelor sistemului.

Ex.

Arhitectura, elementele fizice,
configurația la execuție (runtime),
elementele specifice domeniului aplicației.

UML – diagrame

Reprezintă modul în care
toate elementele
sistemului
colaborează pentru
îndeplinirea
funcționalităților lor
(comportamentul)



UML - diagramele

Diagramele de structură

Diagramele de comportament

Utilizate în conjuncție pentru a ilustra un aspect particular al sistemului.

Clasă

Diagramă de stări :
comportamentul condus de evenimente al instanțelor clasei

Diagramă de obiecte (instanțe)

Diagramă de interacțiune (interacțiune între instanțe)

UML - discuție

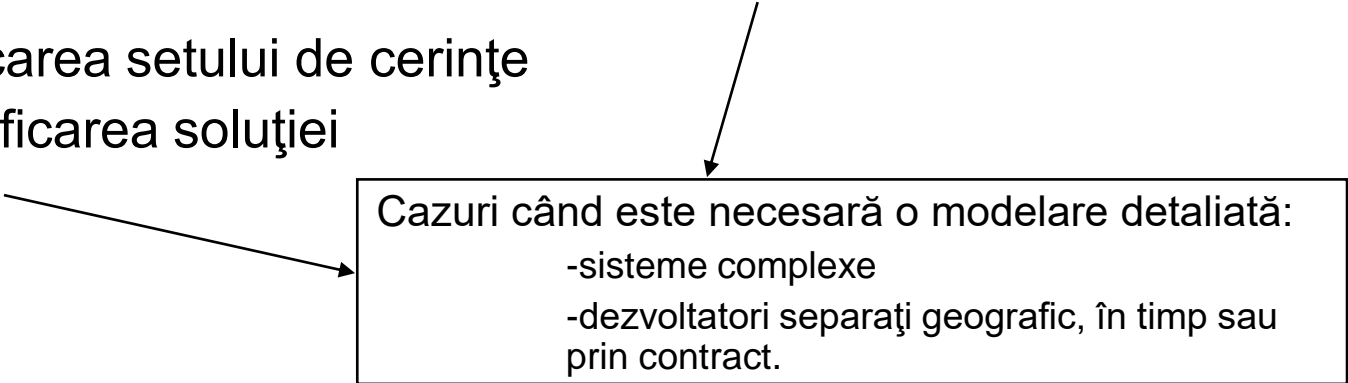
NOTAȚIA – vehicol pentru capturarea unui model, nu un scop în sine.

Recomandare:

- aplicarea doar a acelor elemente necesare comunicării intenției

Pericole:

- subspecificarea setului de cerințe
- supraspecificarea soluției



Cazuri când este necesară o modelare detaliată:

- sisteme complexe
- dezvoltatori separați geografic, în timp sau prin contract.

Alegerea diagramelor utilizate depinde de:

- nivelul de maturitate al modelului: conceptual, logic sau fizic,
- tipul sistemului (ex. diagrame de timp pentru sisteme în timp real).

UML - discuție

- Notăție aplicabilă la o *abordare incrementală, iterativă* de dezvoltare de software.
- Diagramele *evoluează* pe parcursul procesului de proiectare, pe măsură ce se iau noi decizii și se stabilesc mai multe detalii.
- Notăție *independentă* de limbajul de programare.
- *Legături* între diagrame – permit urmărirea inversă a cerințelor de la implementare la specificare.

Ex.

Artefact în nod în *diagramă de instalare*

⇒ Componentă în *diagramă de componente*

⇒ Colecție de clase în *diagramă de clase*

⇒ *Cazuri de utilizare și cerințe.*

PLAN CURS

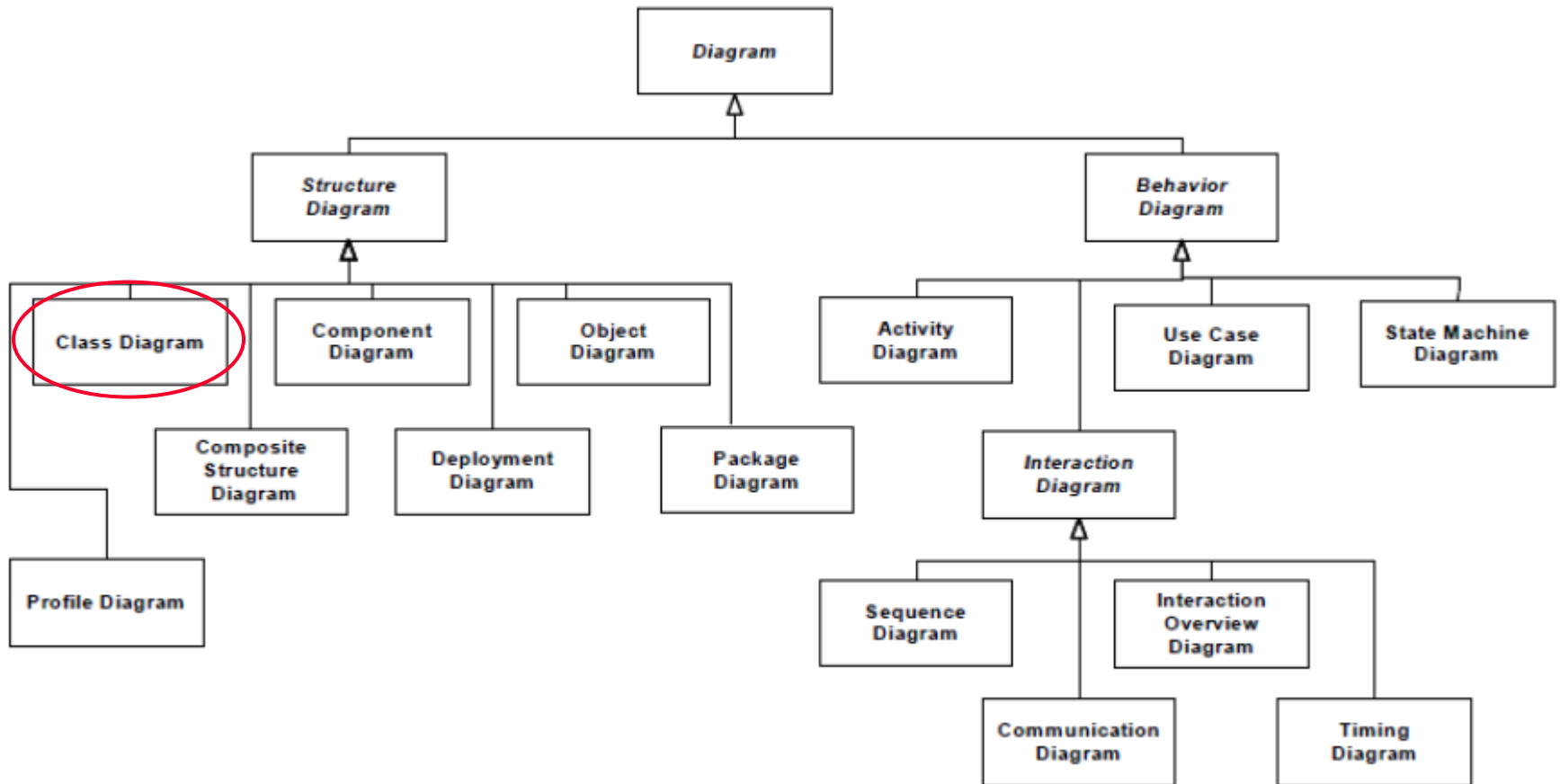


DIAGRAMA DE CLASE

- Modelează resursele necesare construirii și operării unui sistem
- Sursă pentru generare de cod și destinație pentru inginerie inversă
- Modelează fiecare resursă în termeni de structură, relații și comportament.

DIAGRAMA DE CLASE

REPREZENTARE CLASĂ

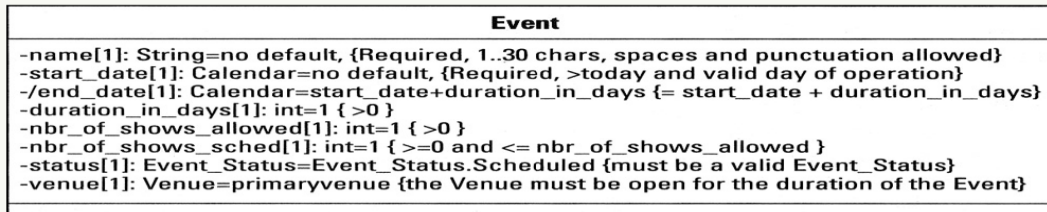
Format specificare nume

[stereotype] name [{property-string}]



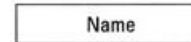
Format specificare atribut

[visibility] [/] name [multiplicity] [: type] [=default] [{property-string}]

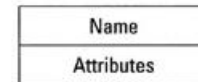


Format specificare operație: [visibility] name ([parameter-list]) [':'] [return-result] [{properties}]
unde: parameter-list:= name [':' data-type] ['[' multiplicity '']]

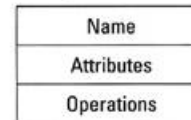
name compartment only - minimum



name and attributes only
operations suppressed



all compartments visible



name and operations only
attributes suppressed

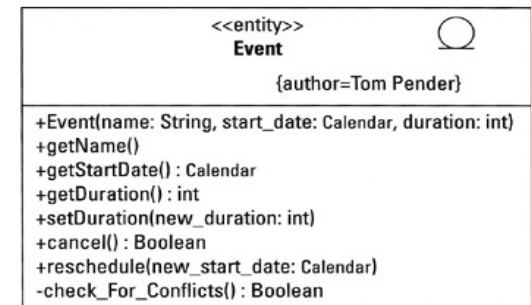
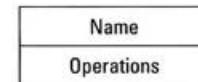
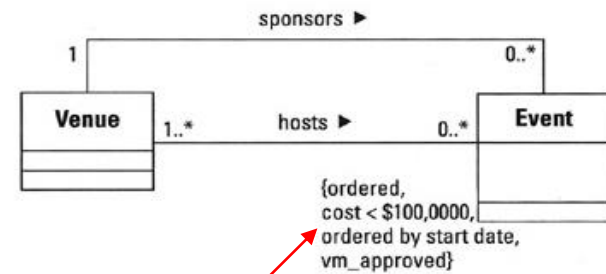
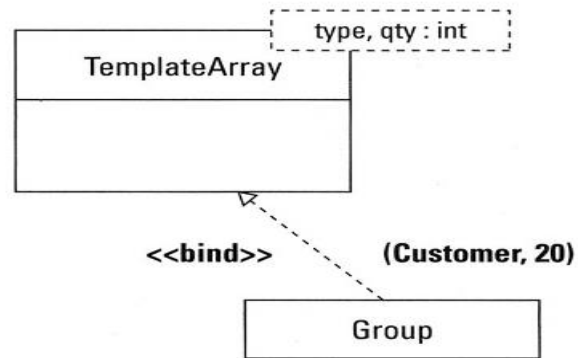


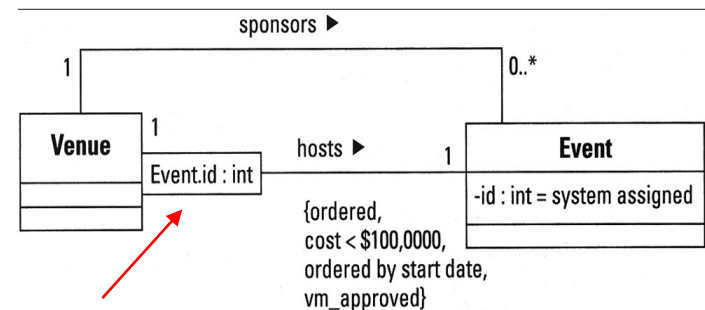
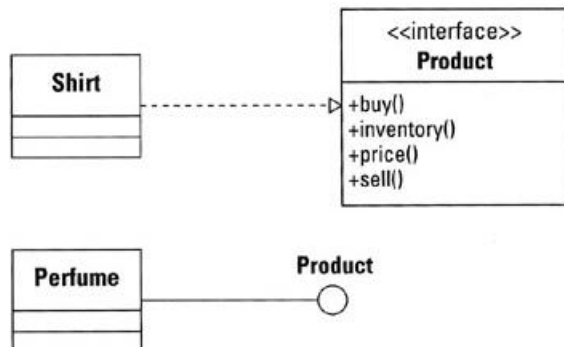
DIAGRAMA DE CLASE

CLASE PARAMETRIZATE (template)



Constraint (OCL)

INTERFEȚE



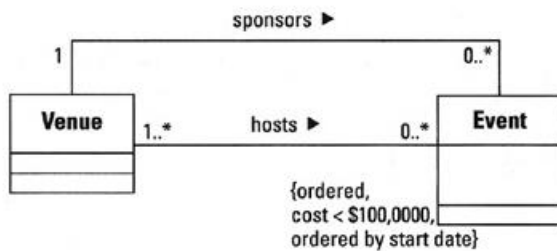
CALIFICATOR:
Atribut cu rol de identificador unic

DIAGRAMA DE CLASE

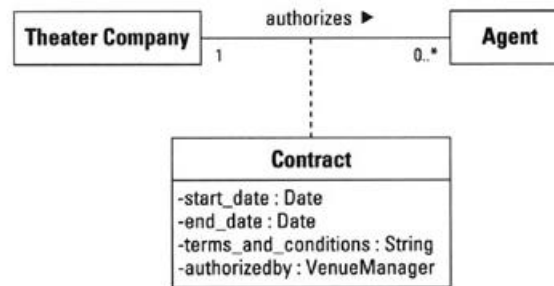
RELAȚII ÎNTRE CLASE

ASOCIERE

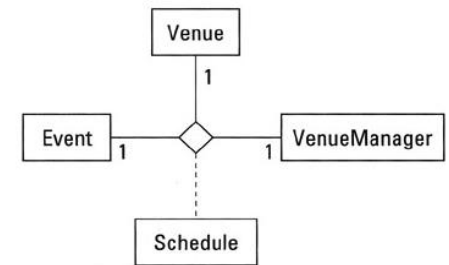
binară



Clasă asociere



Asociere *n*-ară

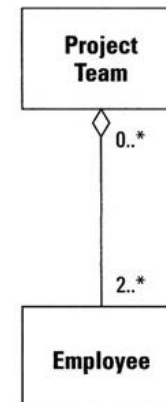


AGREGARE și COMPOZIȚIE - "is part of"

Agregare: obiectele membru au durată de viață independentă de cea a agregatului.

Compoziție: obiectul compus controlează crearea/distrugerea membrilor. (agregare tare)

Aggregation



Composition

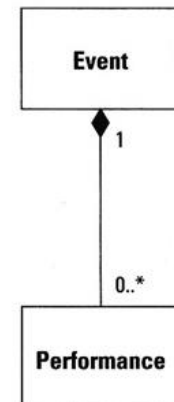
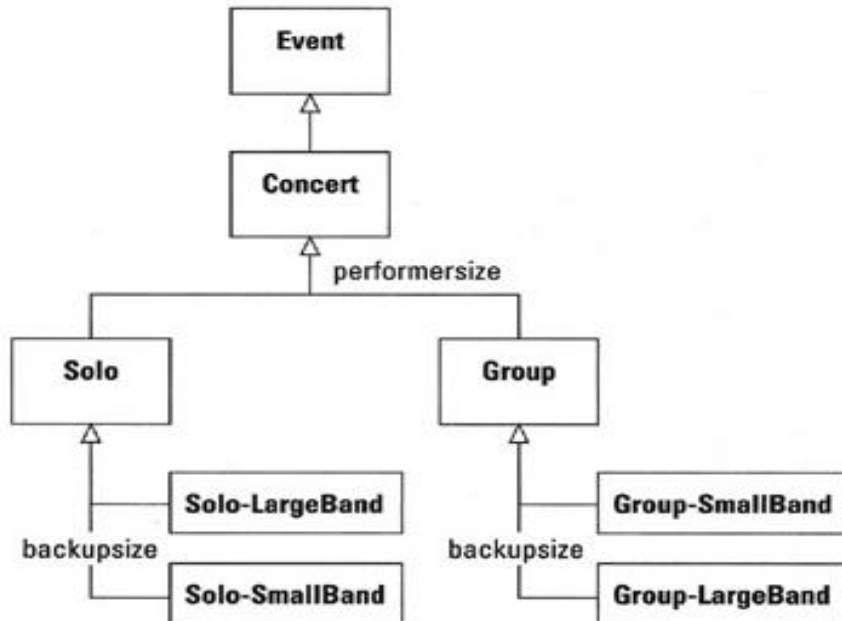
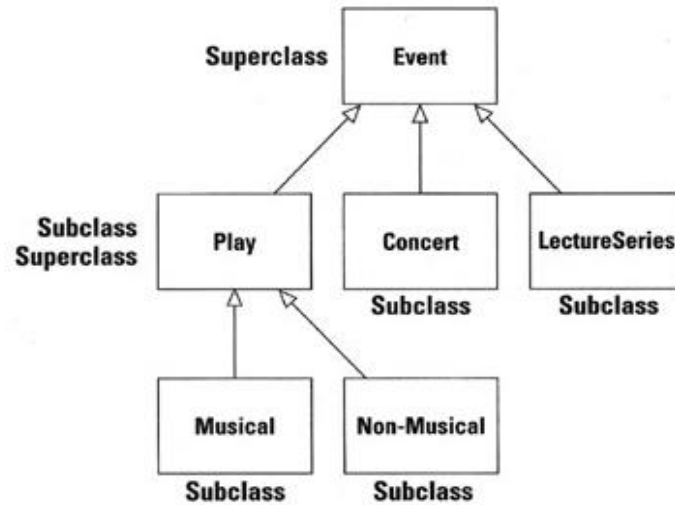


DIAGRAMA DE CLASE

RELAȚII ÎNTRE CLASE

GENERALIZARE/SPECIALIZARE(1)

Definire subclase utilizând discriminator



Variante de modelare clasă abstractă

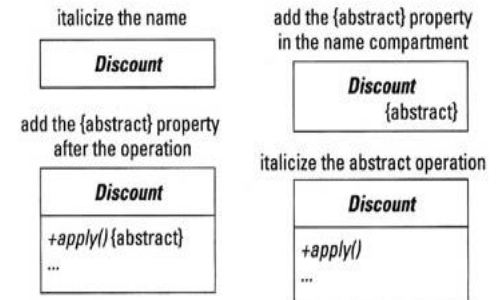
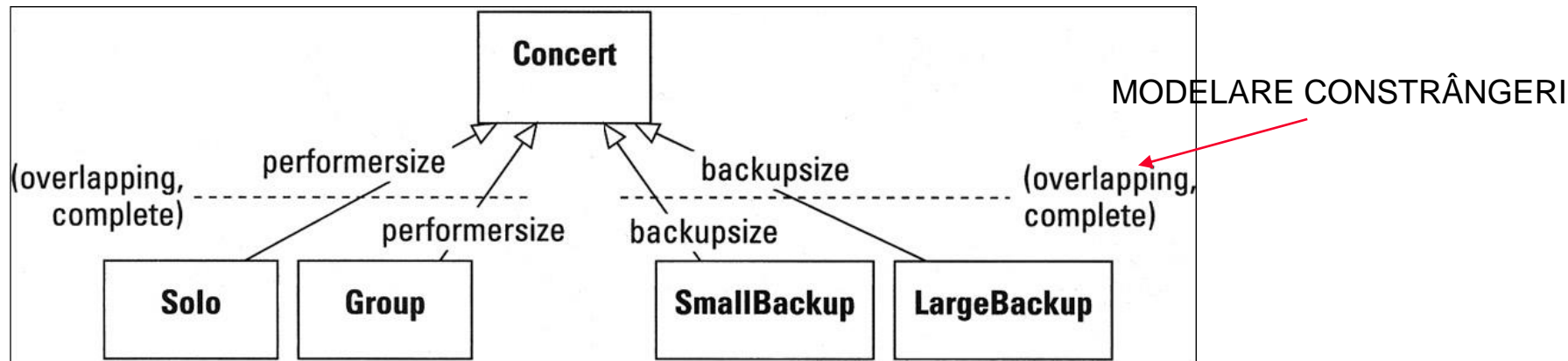


DIAGRAMA DE CLASE

RELAȚII ÎNTRE CLASE

GENERALIZARE/SPECIALIZARE (2)

Grupare generalizări în dimensiuni ortogonale folosind *generalization set*



TIPURI DE CONSTRÂNGERI specifice relației de generalizare/specializare

overlapping – o instanță poate deriva din clase multiple specializate cu discriminatori diferiți

disjoint - not-overlapping

complete – au fost identificate și modelate toate subclasele posibile (nu neaparat vizibile în diagrama curentă)

incomplete – cercetarea asupra posibilelor subclase este în curs.

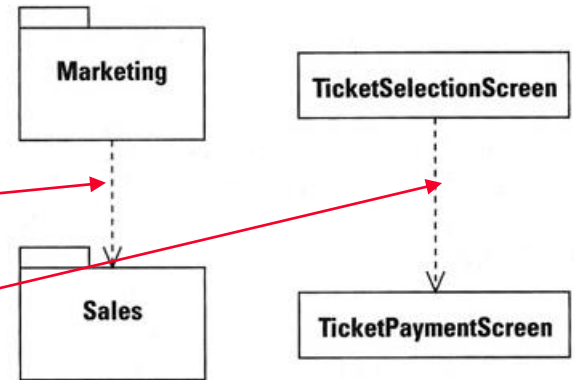
DIAGRAMA DE CLASE

RELAȚII ÎNTRE CLASE

DEPENDENȚĂ - nivel înalt de abstractizare; poate fi definită și între alte tipuri de clasificatori (ex. pachete)

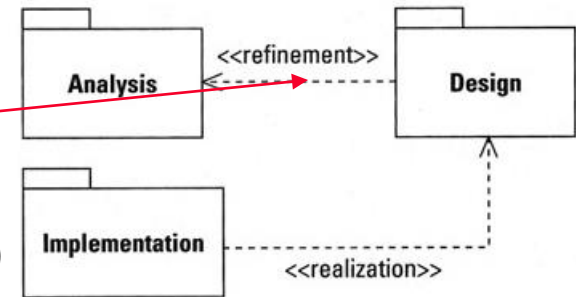
- Relație de tip client-furnizor; o modificare la furnizor necesită modificare la client.

- Relație de precedență.



Stereotipuri pentru relatia de dependență predefinite în UML:

- Permise (`<<access>>`, `<<import>>`)
- Legare (`<<bind>>`)
- Abstractizare (`<<derive>>`, `<<refinement>>`, `<<trace>>`)
- Utilizare (`<<use>>`, `<<call>>`, `<<instantiate>>`, `<<send>>`)
- Substituire (`<<substitute>>`)



PLAN CURS

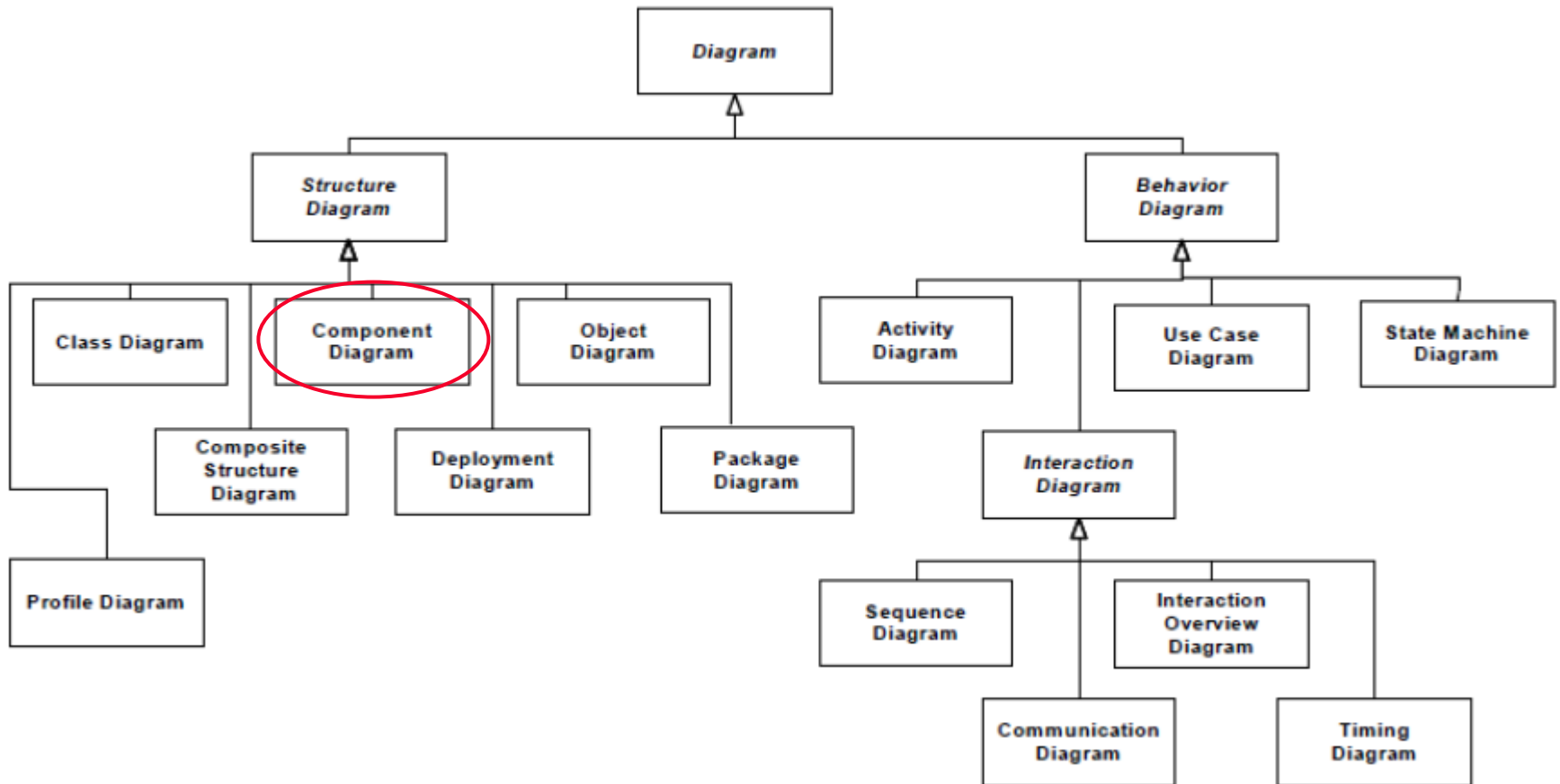


DIAGRAMA DE COMPONENTE

COMPONENTA:

- Element software reutilizabil ce oferă o anumită funcționalitate.
- Cluster de clase coezive și slab cuplate cu exteriorul
- O clasă poate fi conținută într-o singură componentă
- O componentă poate conține componente
- Colaborează cu alte componente prin interfețe bine definite

Utilizate în descompunerea ierarhică a sistemului și în reprezentarea arhitecturii logice.

DIAGRAMA DE COMPONENTE

DIAGRAMA: elementele esențiale:

- *Componentele*
- *Interfețele* componentelor: definesc detaliile de interacțiune
- *Realizările* componentelor

DIAGRAMA DE COMPONENTE

Notația pentru componentă: (perspectivă “black box”)

Port: vizibilitate publică

Identificator (opțional):

nume_port :

[tip_port]

**Interfață solicitată
(required)**

**Interfață oferită
(provided)**

Obs. La un port se pot grupa mai multe interfețe.

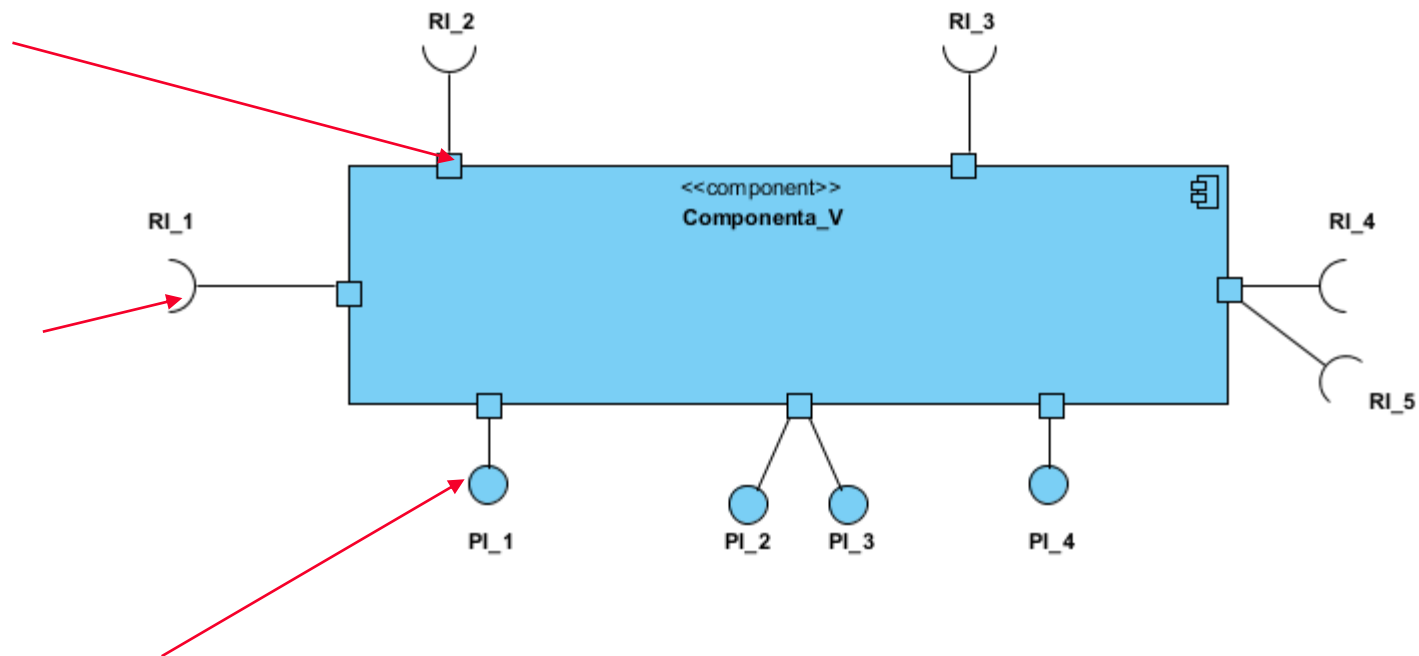


DIAGRAMA DE COMPONENTE

DIAGRAMA reprezintă:

- nivelele logice și partiționarea arhitecturii
- interdependențele componentelor

Conector de asamblare/de interfață

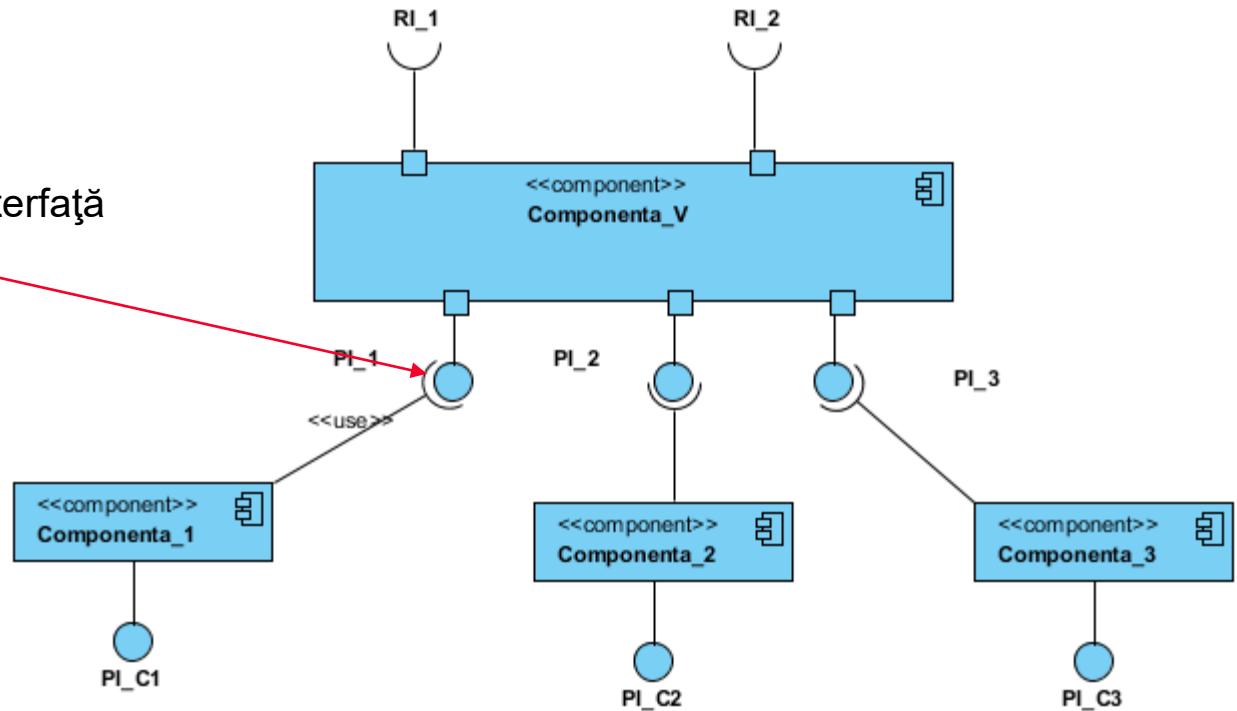
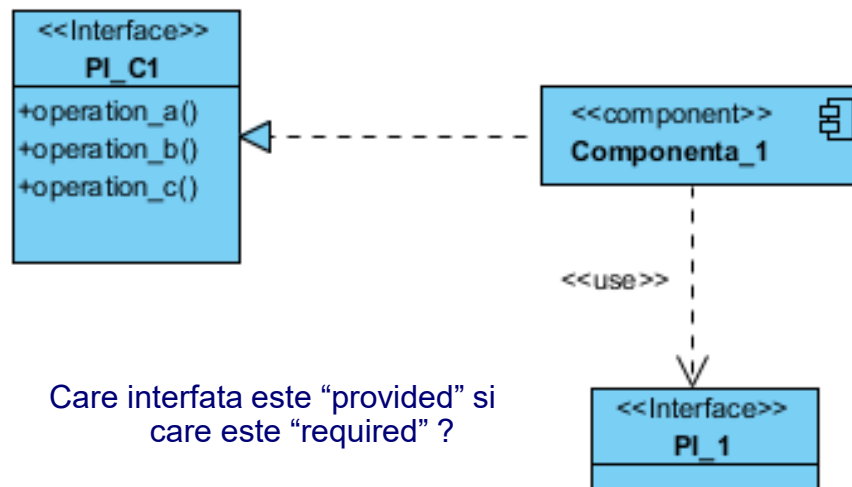


DIAGRAMA DE COMPONENTE

DIAGRAMA: detalii de specificare interfețe (notații alternative)



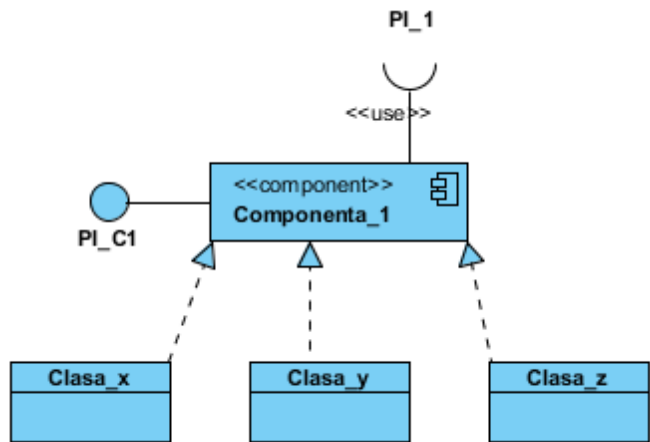
Care interfata este “provided” si
care este “required” ?

Reprezentare parțială

DIAGRAMA DE COMPONENTE

Relația de tip “realizează”.

O componentă e realizată de un set de clase.



Reprezentări alternative

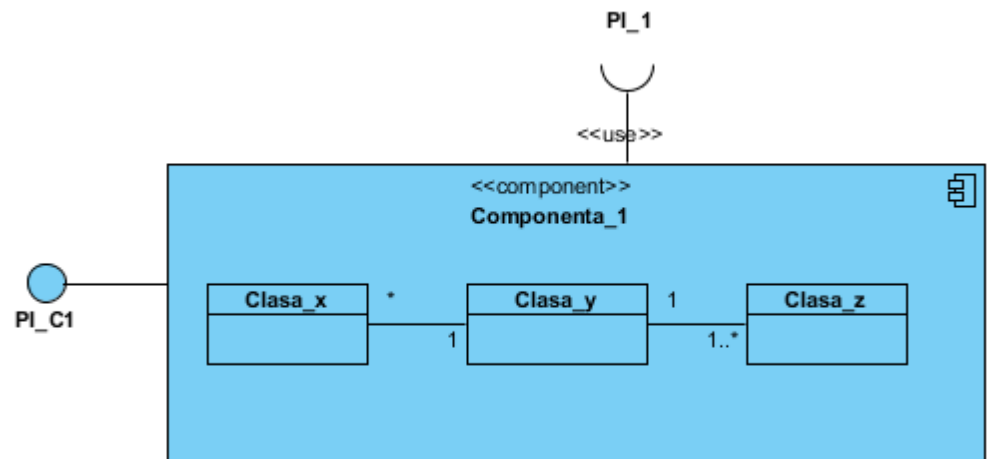
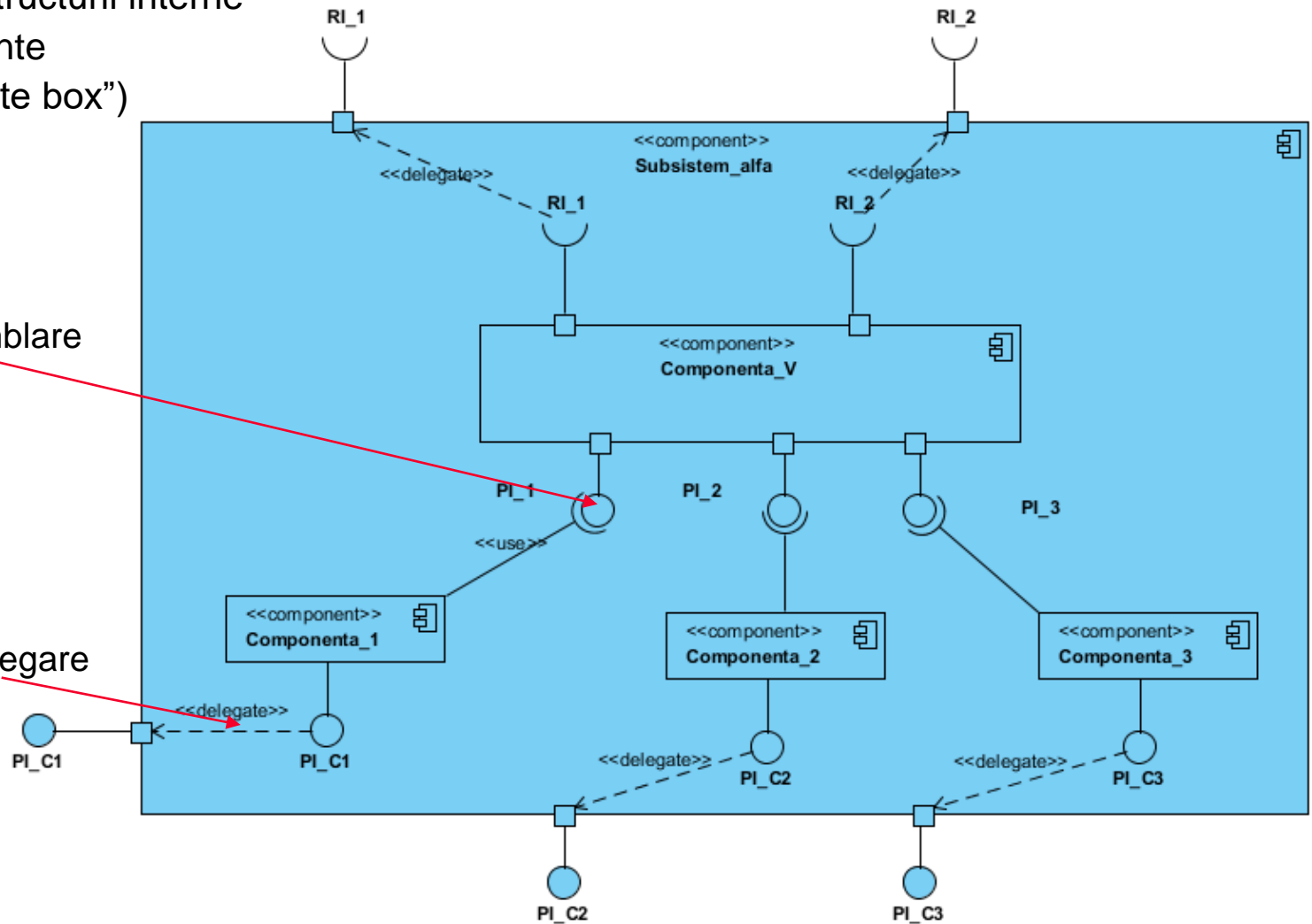


DIAGRAMA DE COMPONENTE

Reprezentarea structurii interne
a unei componente
(perspectivă “white box”)

Conector de asamblare

Conector de delegare



PLAN CURS

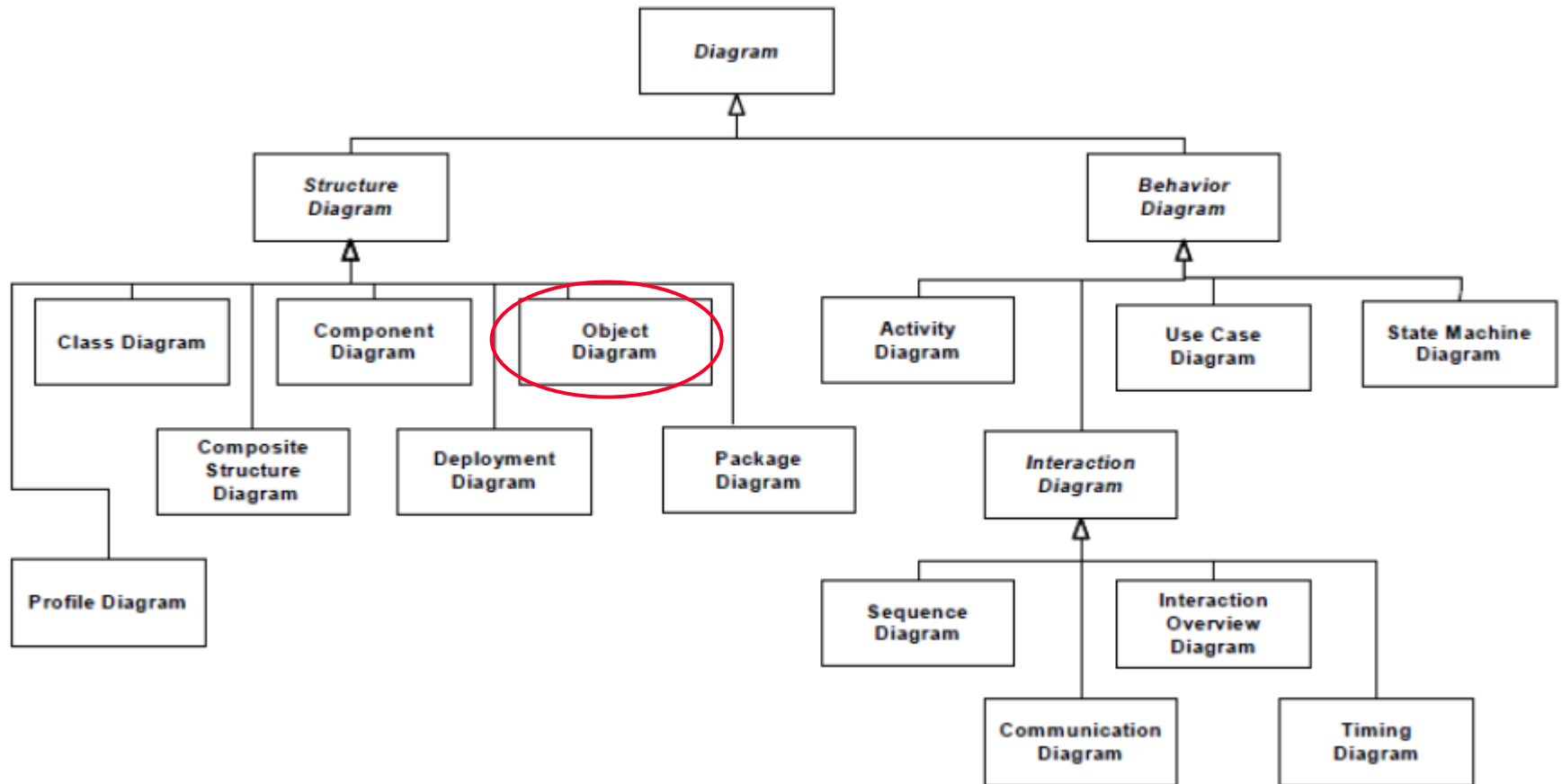


DIAGRAMA DE OBIECTE

Instrument UML pentru modelare de exemple - exemple concrete pentru diagramele de clase asociate.

Diagrama de clase modelează definiții și reguli.

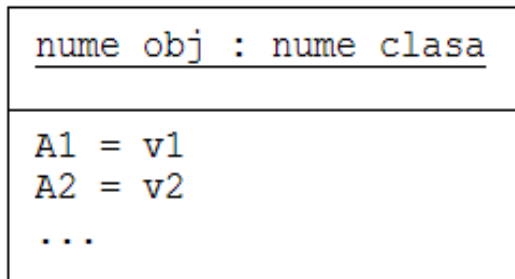
Diagrama de obiecte modelează fapte.

Perspectivă instantanee asupra unei configurații de obiecte.

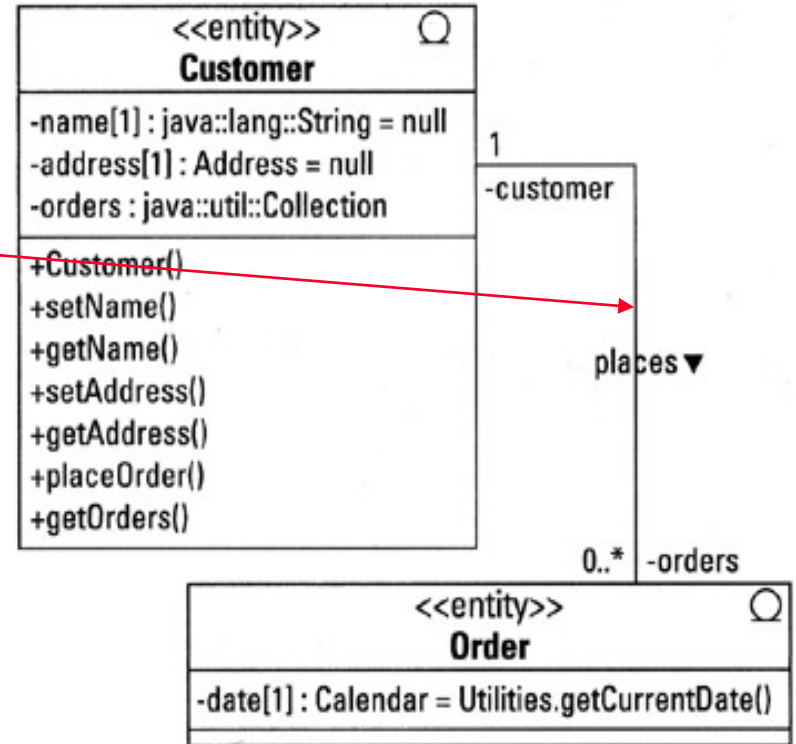
DIAGRAMA DE OBIECTE

DIAGRAMĂ CLASE

Reprezentare obiect

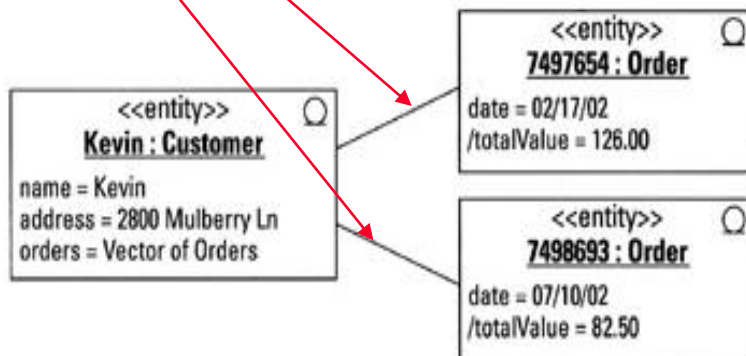


Asociere



Link = instanță
a asocierii

DIAGRAMĂ OBIECTE



PLAN CURS

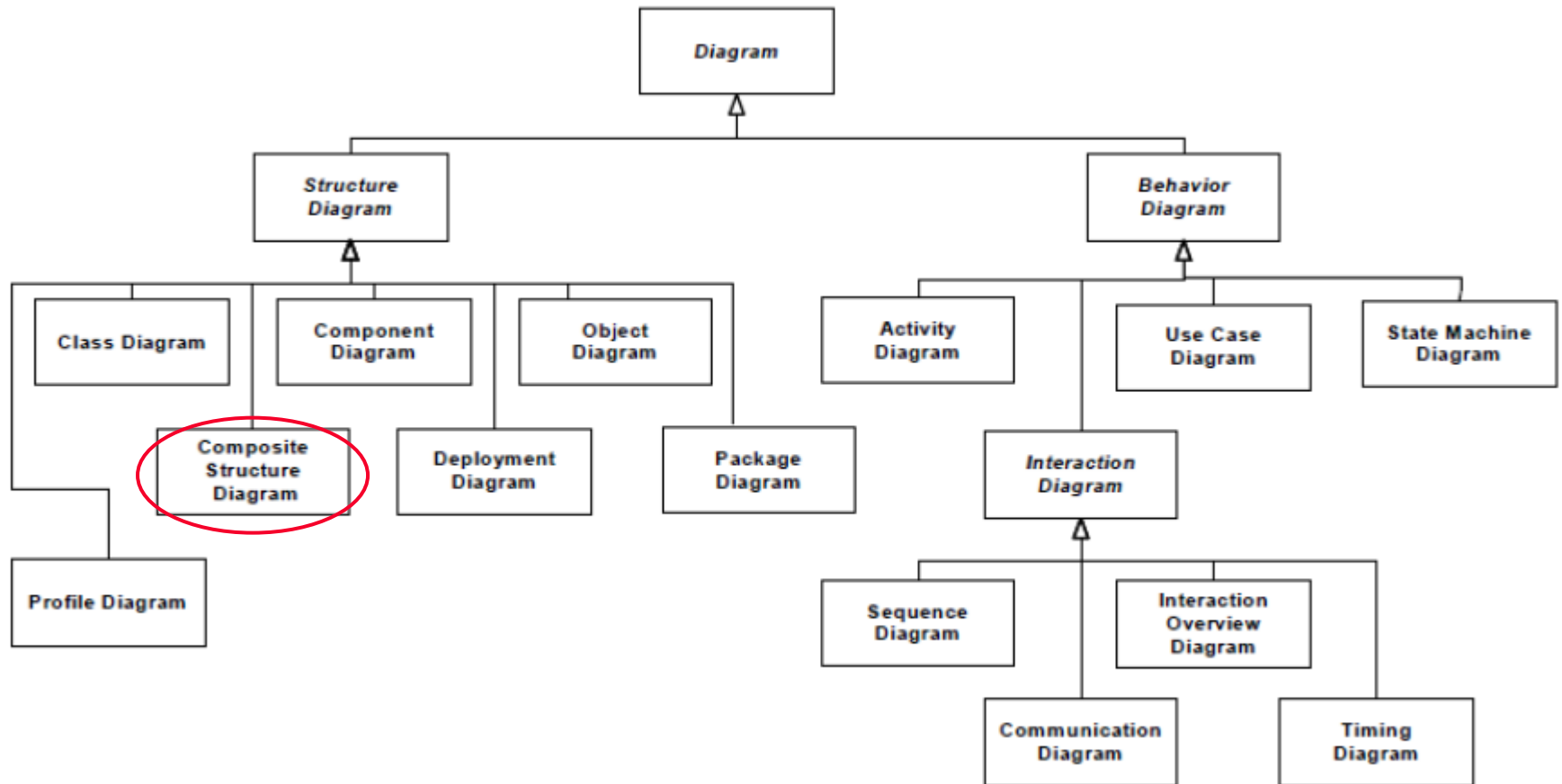


DIAGRAMA STRUCTURII COMPUSE

Diagrama structurii compuse - reprezintă *structura internă* a unui clasificator structurat și *punctele de interacțiune* ale acestuia cu restul sistemului.

Def. Clasificator — un tip de element modelat care poate avea instanțe (e.g., o clasă este un clasificator ale cărei instanțe sunt obiecte; un tip de componentă este un clasificator ale cărui instanțe sunt componente de tipul respectiv).

Clasificator structurat – este definit atât în ansamblu cât și pentru fiecare parte a sa în termeni de INSTANȚE conținute (proprie sau referite de clasificatorul structurat).

Parte - Instanță conținută care este proprie

cu semnificație similară cu a relației de compoziție

- Creată în cadrul clasificatorului ce o conține
- Distrusă odată cu distrugerea clasificatorului ce o conține.
- Poate conține, la rândul ei, structuri compuse

DIAGRAMA STRUCTURII COMPUSE

Concepte

Parte – **rol** jucat la execuție de una sau mai multe instanțe ale clasificatorului structurat.

Port – punct de interacțiune pentru conexiunea între părți sau cu exteriorul.

Poate specifica servicii oferite (provided) sau solicitate (required).

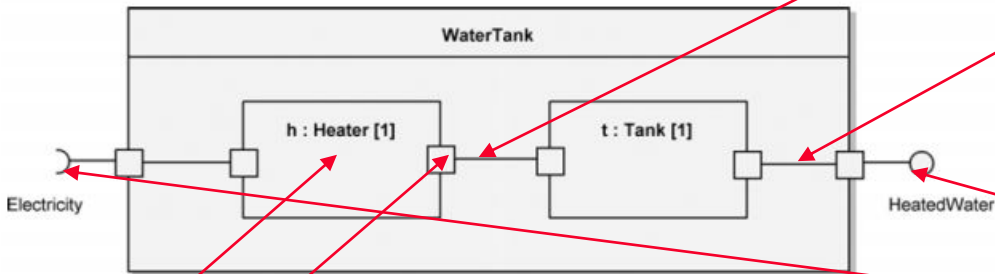
Conector – leagă două sau mai multe entități ce interacționează la execuție.

Colaborare – definește un set de elemente participante care cooperează, în termeni de roluri, pentru realizarea unei anumite sarcini.

DIAGRAMA STRUCTURII COMPUSE

Tip clasificator structurat: **COMPONENTĂ**

ELEMENTE DE REPREZENTARE



Conector

.de asamblare (de interfață)
.de delegare

Format nume: [nume_conector]

Interfață

.oferită
.solicitată

Parte componentă

Format nume: nume_rol : nume_clasă [multiplicitate]

Port

Format nume: nume_port [: tip_port [multiplicitate]]

DIAGRAMA STRUCTURII COMPUSE

Tip clasificator structurat: COLABORARE

Def. Tip de clasificator structurat care specifică *interacțiunile*, la momentul execuției, între *istanțe de clasificatori*.

Definește un set de elemente participante care cooperează pentru realizarea unei anumite sarcini în termeni de roluri.

- Definește rolurile asumate de instanțe și conectorii prin care colaborează pentru a oferi o anumită funcționalitate.
- Poate include o altă colaborare.
- Detaliile colaborării pot fi exprimate cu o diagramă de interacțiune.

Exemple de:

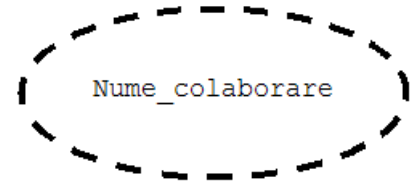
Elemente participante la colaborare : clase și obiecte, asocieri și legături, attribute și operații, interfețe, cazuri de utilizare, componente, noduri.

Comportament descris de colaborare : caz de utilizare, operație, colecție de operații, mecanism general al sistemului.

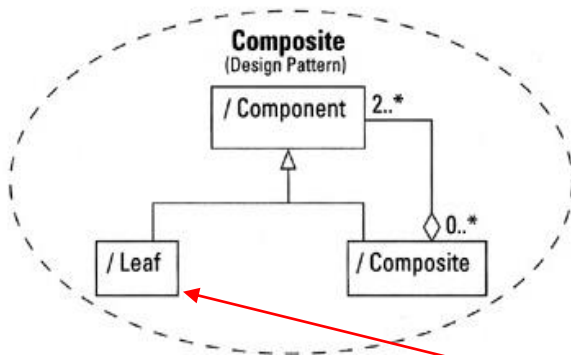
DIAGRAMA STRUCTURII COMPUSE

COLABORARE

Reprezentare generală

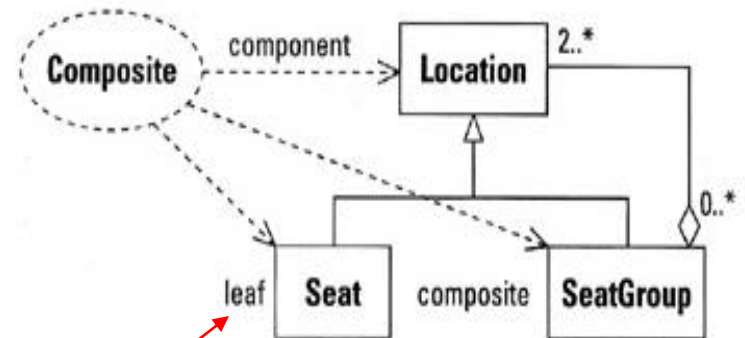


Modelarea unui șablon de proiectare :
roluri legate prin conectori

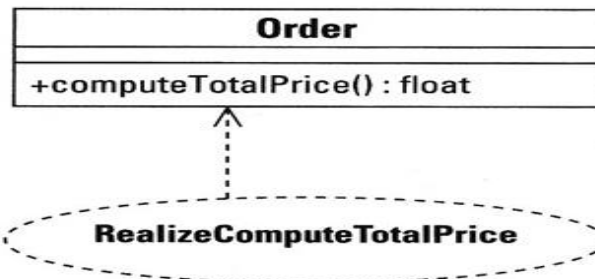


Atașarea la un set
specific de elemente
(clase și relații)

O utilizare a colaborării :
roluri legate la instanțe de clasificatori



Modelarea unei colaborări
ce realizează o operație



ROL – definește proprietățile pe care un
clasificator structurat trebuie să le aibă
pentru a participa în colaborare.

Un anumit clasificator poate implementa interfețe
multiple, care să-i permită să joace roluri multiple
într-o colaborare sau să participe în diferite roluri
la diferite colaborări în același timp.

PLAN CURS

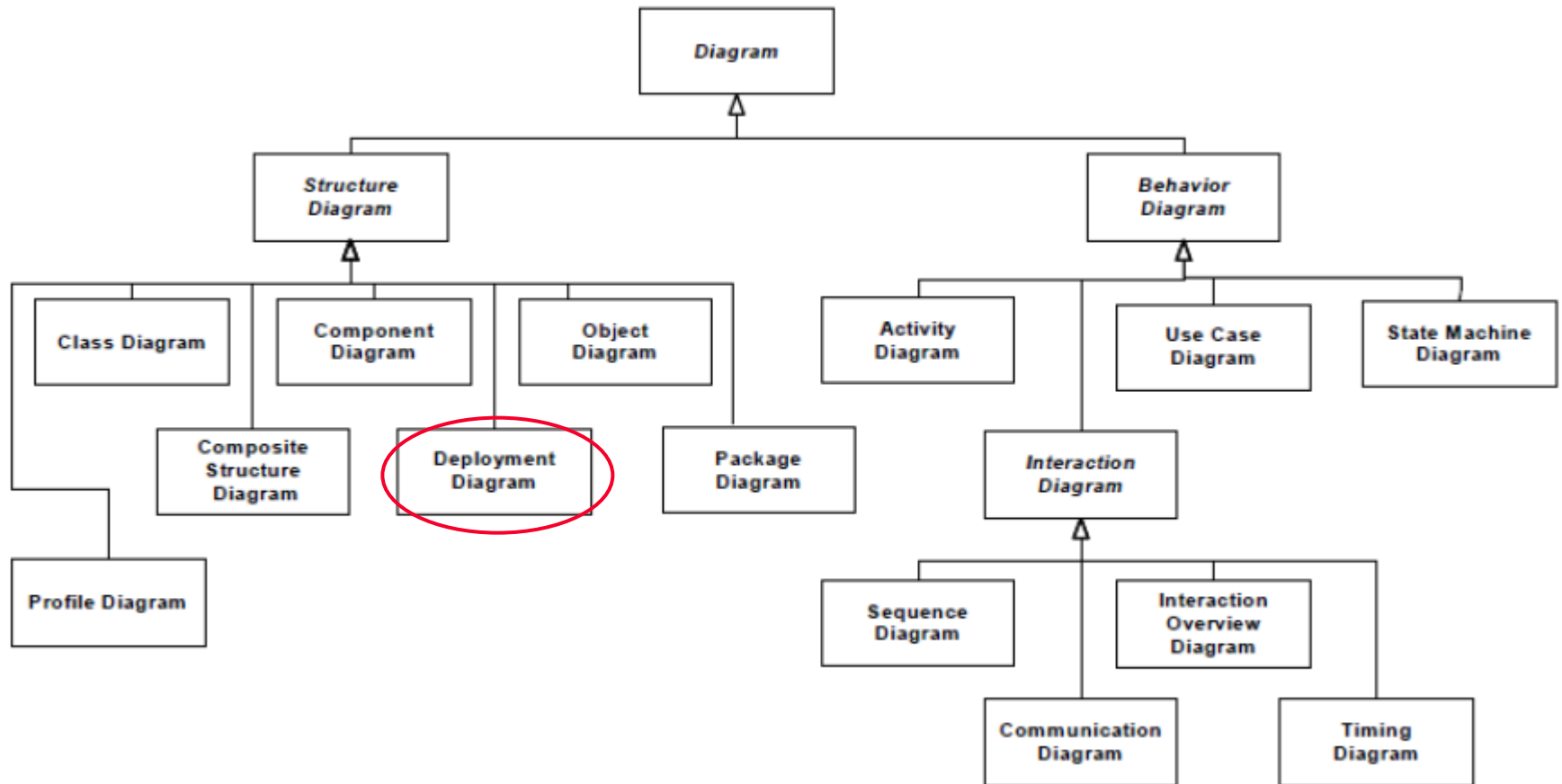


DIAGRAMA DE INSTALARE (REPARTIZARE / DESFĂȘURARE)

Reprezintă *alocarea artefactelor la noduri*, în reprezentarea fizică a unui sistem.

- Perspectivă asupra *structurii* artefactelor sistemului.
- Platforma de execuție a sistemului – colecție de noduri
- Elementele esențiale:
 - Artefactele
 - Nodurile
 - Conexiunile

DIAGRAMA DE INSTALARE

ARTEFACT

- Entitate fizică ce *implementează o porțiune a modelului proiect* pentru software (cod executabil, cod sursă, document, etc.)
- Poate fi în *relație* de dependență sau compoziție *cu alte artefacte*
- Este în *relație* de dependență de tip <<manifest>> cu una sau mai multe *componente*.

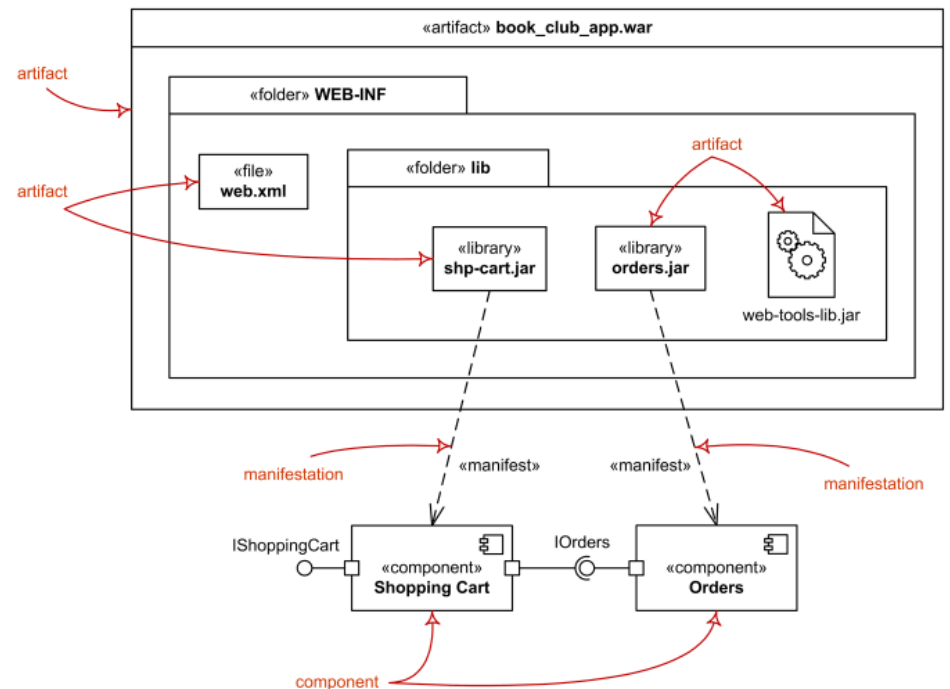
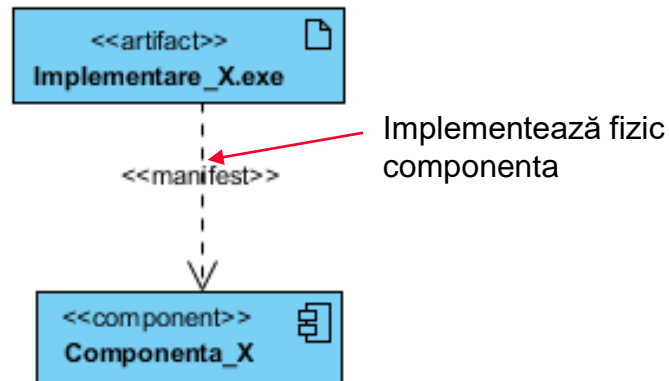


DIAGRAMA DE INSTALARE

NOD

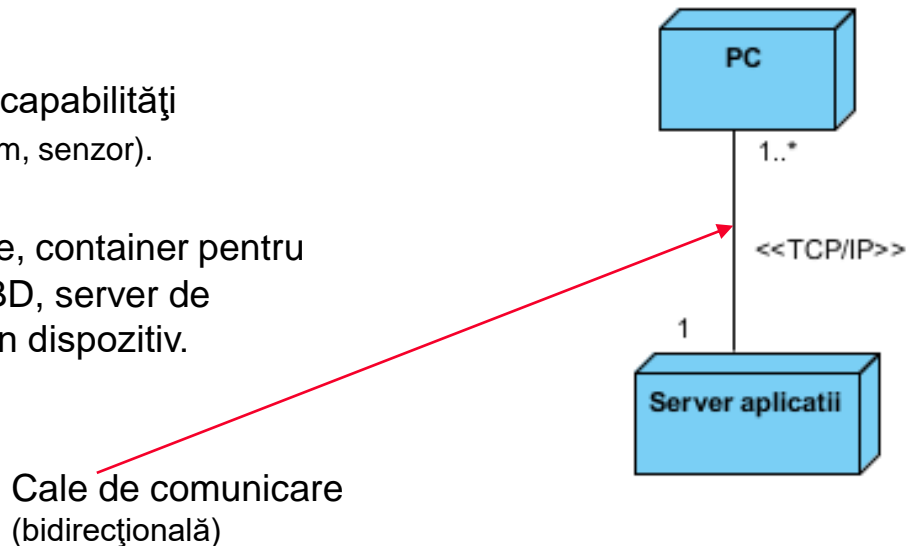
- Resursă computațională (tipic: procesare și/sau memorare) pe care se instalează artefactele în vederea execuției.
- Poate include noduri.

Tipuri de noduri:

Dispozitiv : hardware ce oferă capacități computaționale (ex. calculator, modem, senzor).

Mediu de execuție : software, container pentru diferite tipuri de artefacte (ex. SGBD, server de aplicații); găzduit în mod tipic de un dispozitiv.

Ex. Diagramă cu două noduri.



Note, that **components** were directly deployed to nodes in UML 1.x deployment diagrams. In UML 2.x **artifacts** are deployed to nodes, and artifacts could **manifest** (implement) components. Components are deployed to nodes indirectly through artifacts.

DIAGRAMA DE INSTALARE

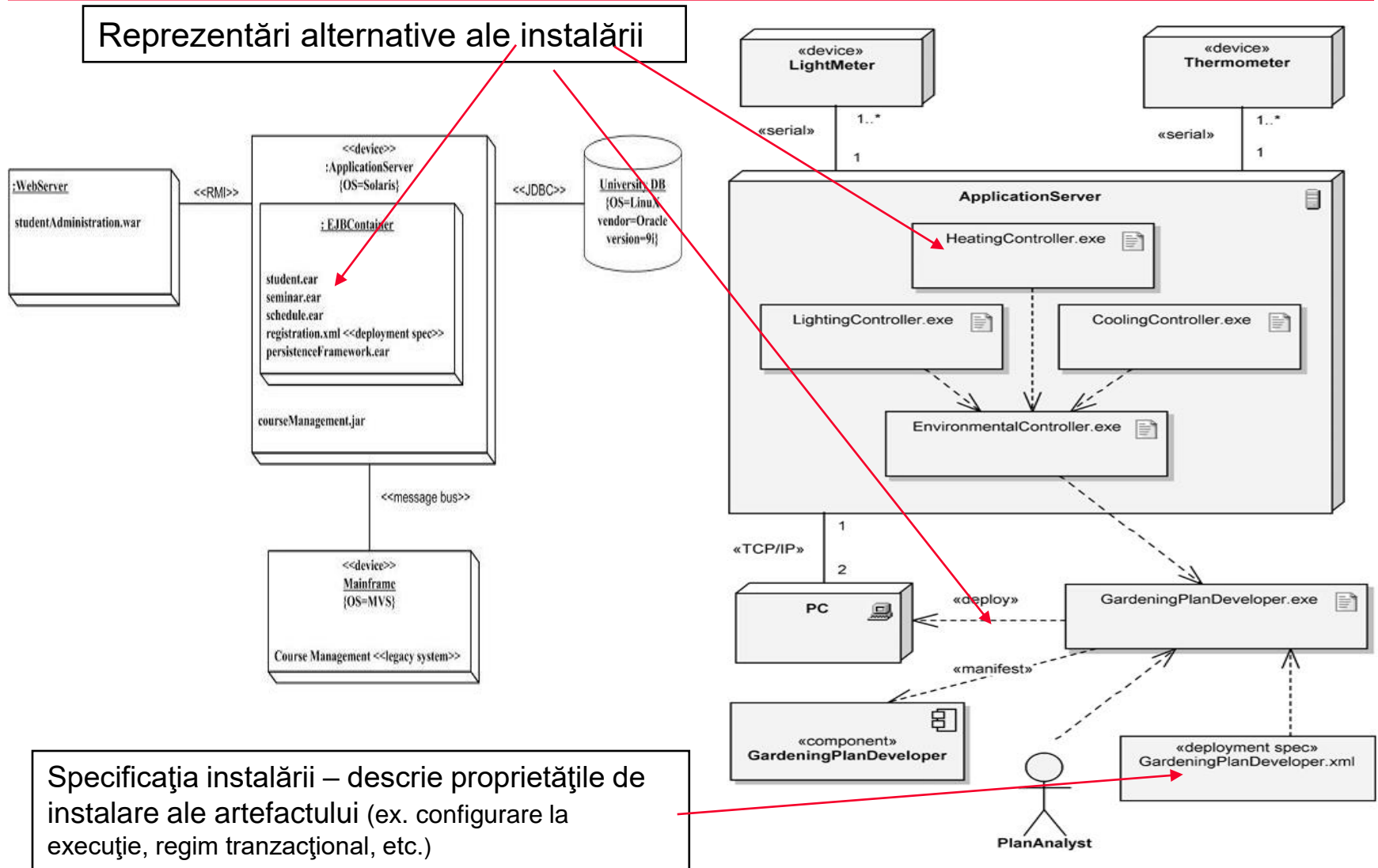
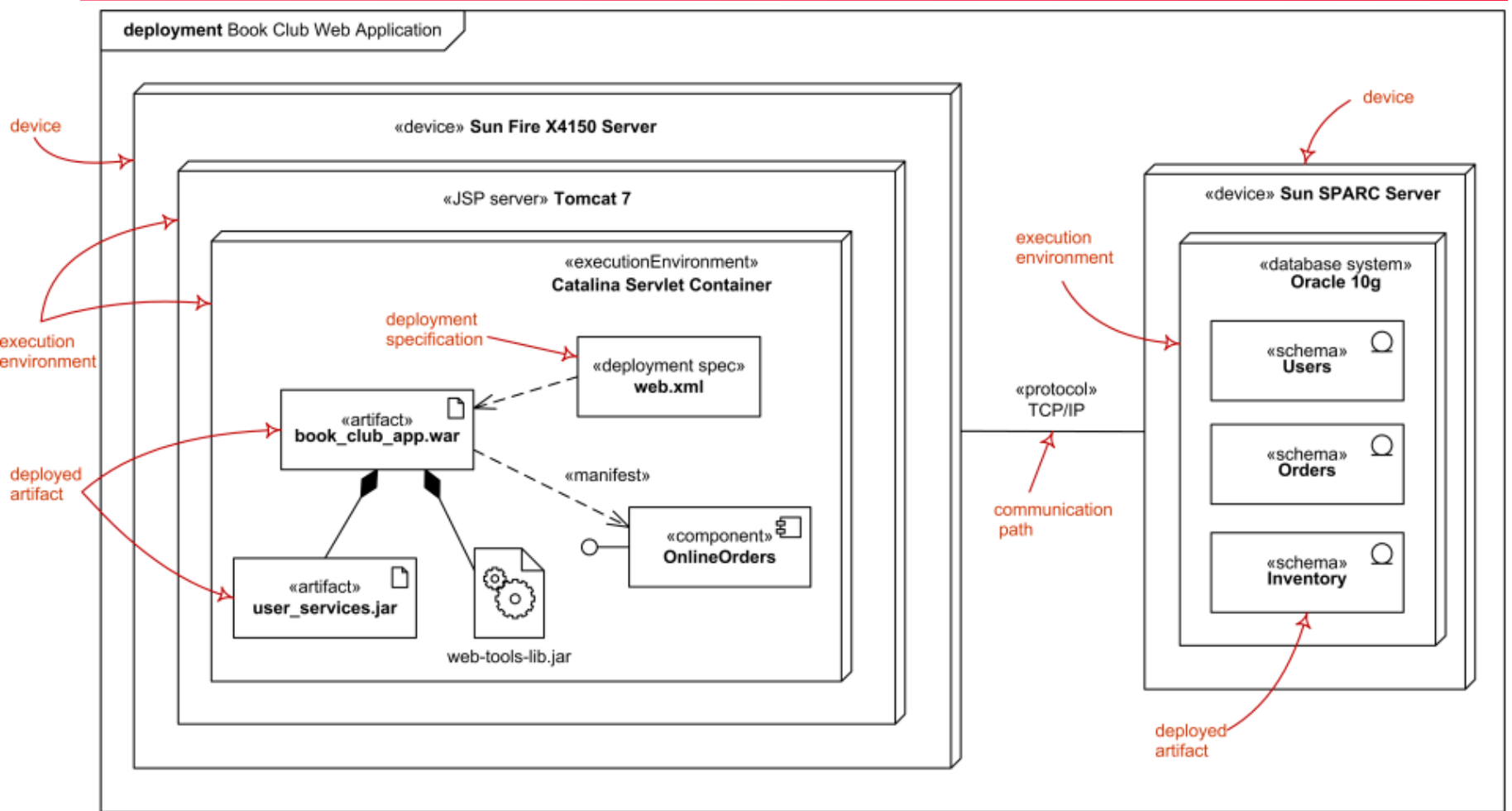


DIAGRAMA DE INSTALARE



Aplicație web instalată pe server JSP Tomcat JSP și schema bazei de date instalată pe SGBD.

PLAN CURS

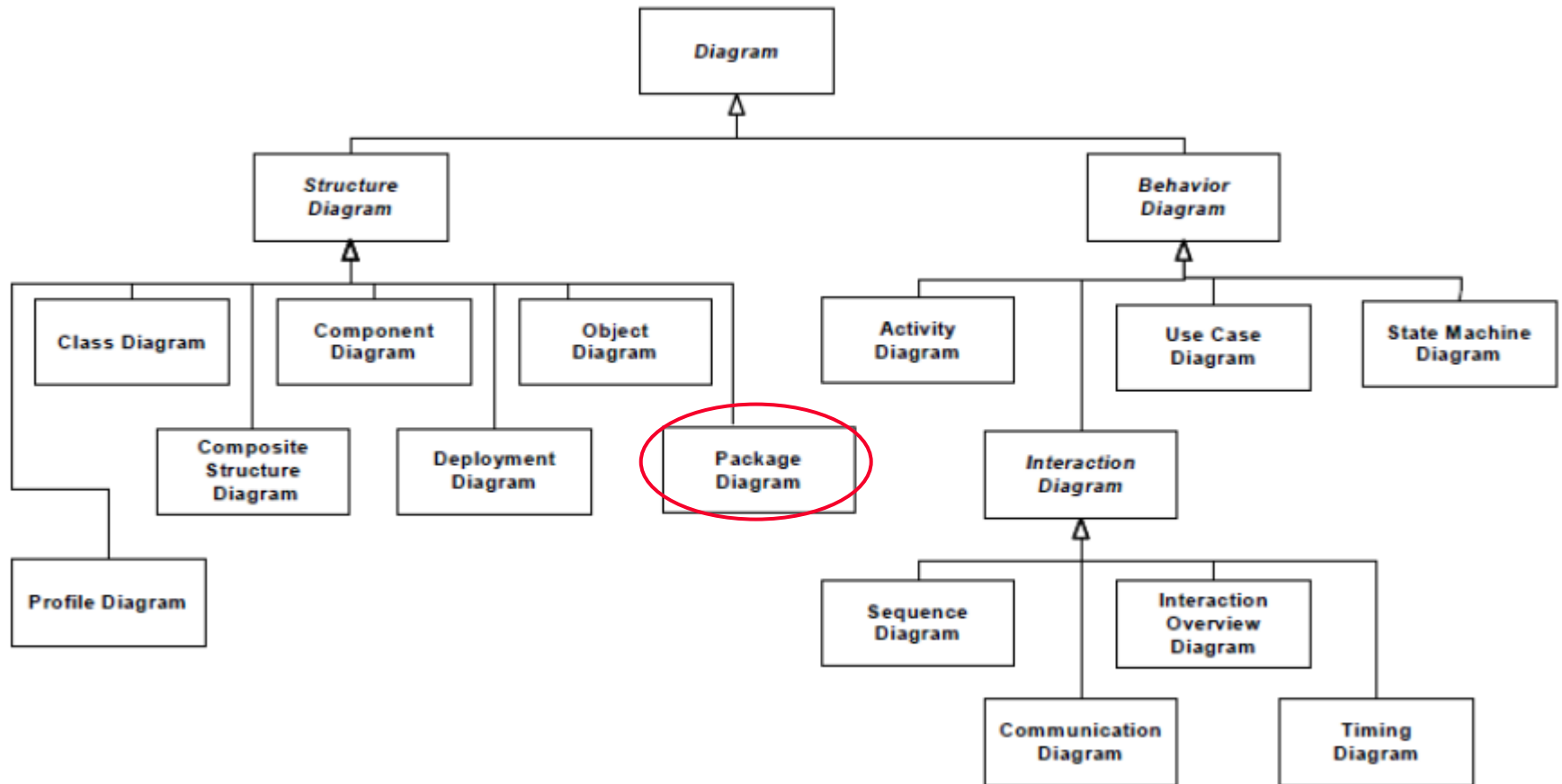


DIAGRAMA DE PACHETE

Diagramă de structură ce conține pachete ca elemente UML primare și arată dependențele între pachete.

Utilizare: ca ***instrument de gestionare a modelelor***

Organizarea artefactelor procesului de modelare, pentru prezentarea clară a spațiului problemei (analiză) și a design-ului asociat (proiect).

Avantaje:

- Claritate în dezvoltarea sistemelor complexe
- Utilizarea concurentă a modelelor
- Controlul versiunilor
- Abstractizare pe nivele multiple
- Încapsulare, modularizare

Grupează reprezentările mai multor elemente UML.

DIAGRAMA DE PACHETE

Elementele esențiale:

- Pachete
- Vizibilitate
- Dependențe

Reprezentare pachet:

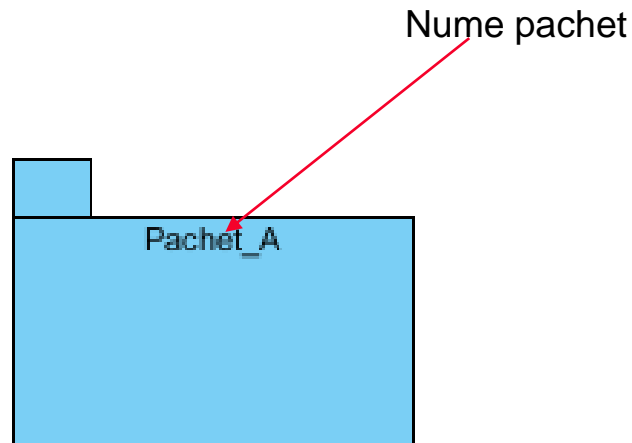


DIAGRAMA DE PACHETE

Reprezentări alternative ale relației “conținut în”:

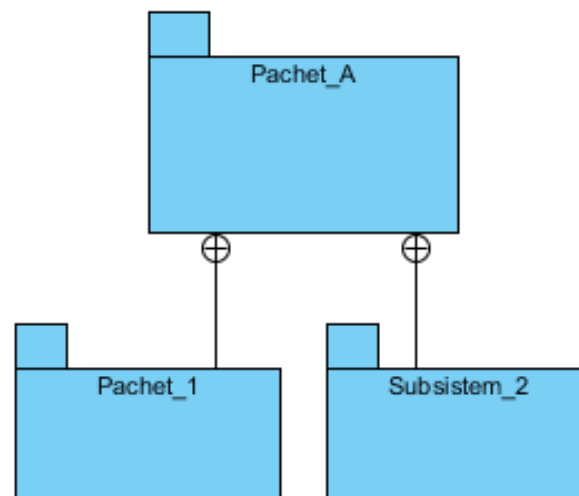
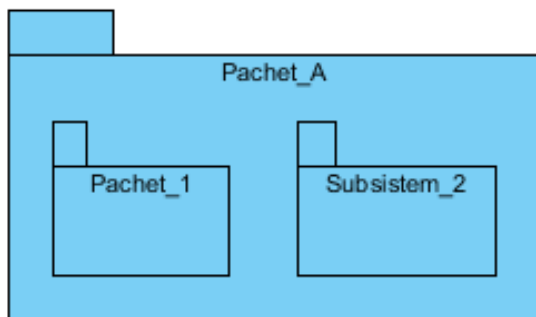


DIAGRAMA DE PACHETE

Vizibilitatea

- definită la nivel de pachet
- aplicată elementelor *conținute* și celor *importate*.

Valori:

- Public (+) : elementele ce fac parte din interfața pachetului
- Private (-) : vizibile doar elementelor din interiorul pachetului (inclusiv pachetelor incluse)

Pachetul oferă un *spațiu de nume* pentru elementele conținute.

- Nume unice în interiorul pachetului pentru elementele de același tip.
- Nume global : `package_name::element_name`

DIAGRAMA DE PACHETE

Relația de dependență (în scopul îndeplinirii unei responsabilități a entității dependente)

Între pachete: `import`, `access`, `merge`

Relativ la elementele conținute: `trace`, `derive`, `refine`, `permit`,
`use`.

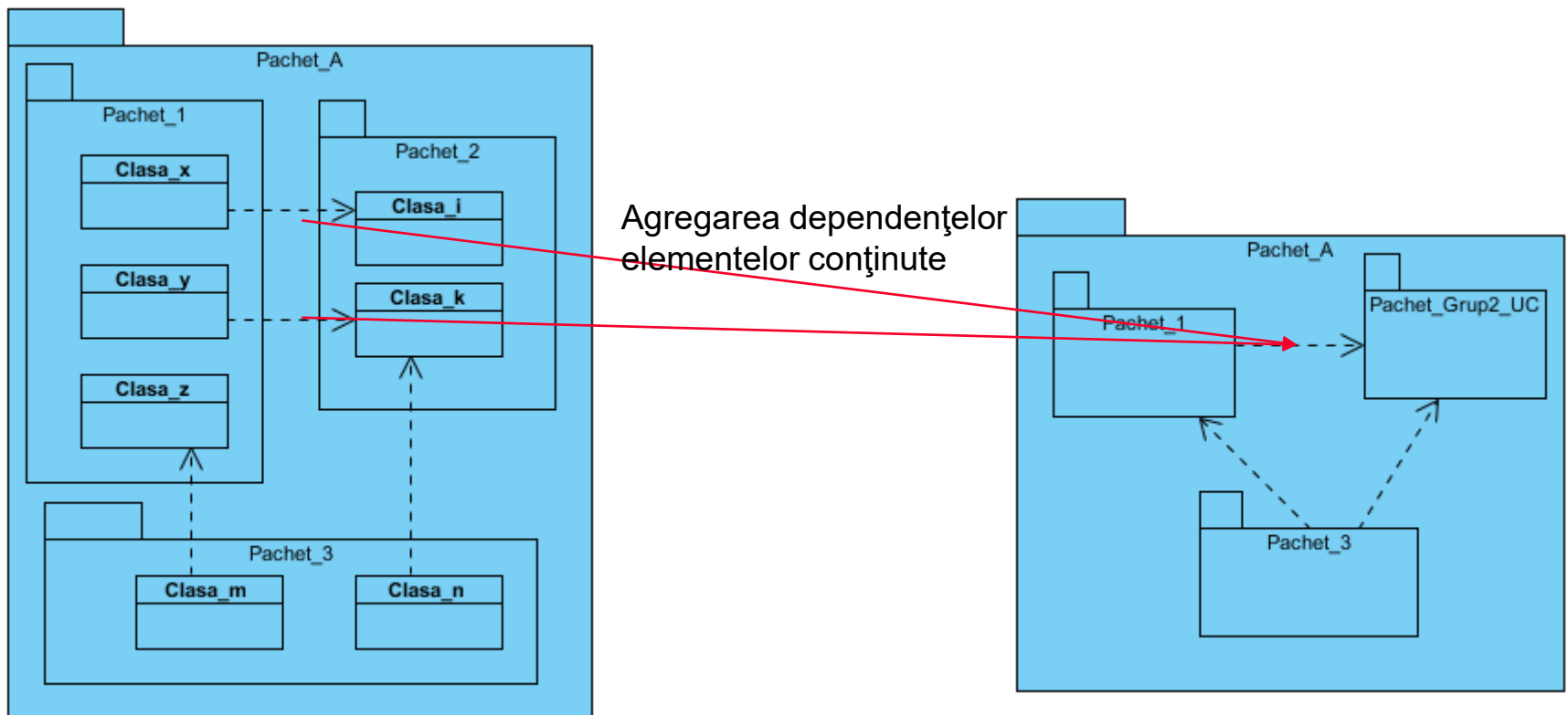


DIAGRAMA DE PACHETE

Elemente conținute în pachet:

- diverse diagrame UML
(nu numai diagrame de structură).

Scop

- clarificare sistem
- partiționare activități de dezvoltare

Grupate pe baza unei legături.

Exemple :

- subsistemele unui sistem
- cazurile de utilizare relative la un anumit aspect al sistemului
- clase ce colaborează la un subset (re)utilizabil de funcționalitate sistem

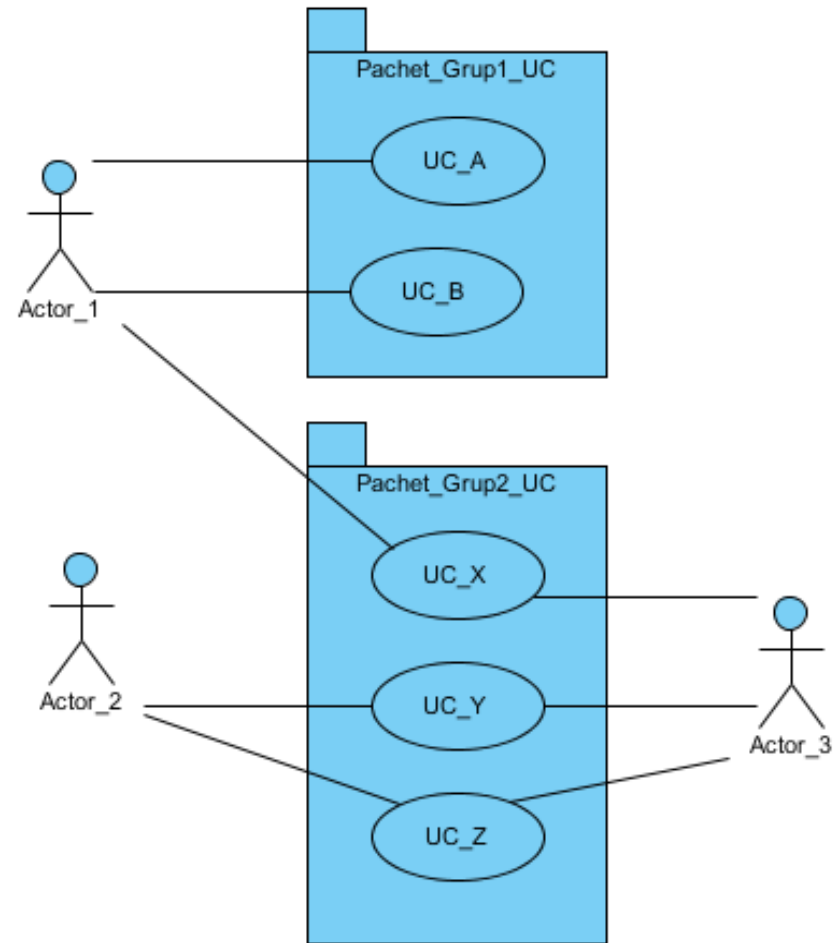


DIAGRAMA DE PACHETE

Deziderate:

- *Coeziune strânsă* a elementelor din același pachet
- *Cuplare slabă* între pachete

Soluție - în același pachet :

- ierarhie de clase
- agregare/compoziție de clase
- cazuri de utilizare aflate în relație.

Reprezintă *structura statică*, indiferentă de timp, a elementelor sistemului.

Ex.

Arhitectura, elementele fizice, configurația la execuție (runtime), elementele specifice domeniului aplicației.

PLAN CURS

Reprezintă modul în care toate elementele sistemului colaborează pentru îndeplinirea funcționalităților lor (*comportamentul*)

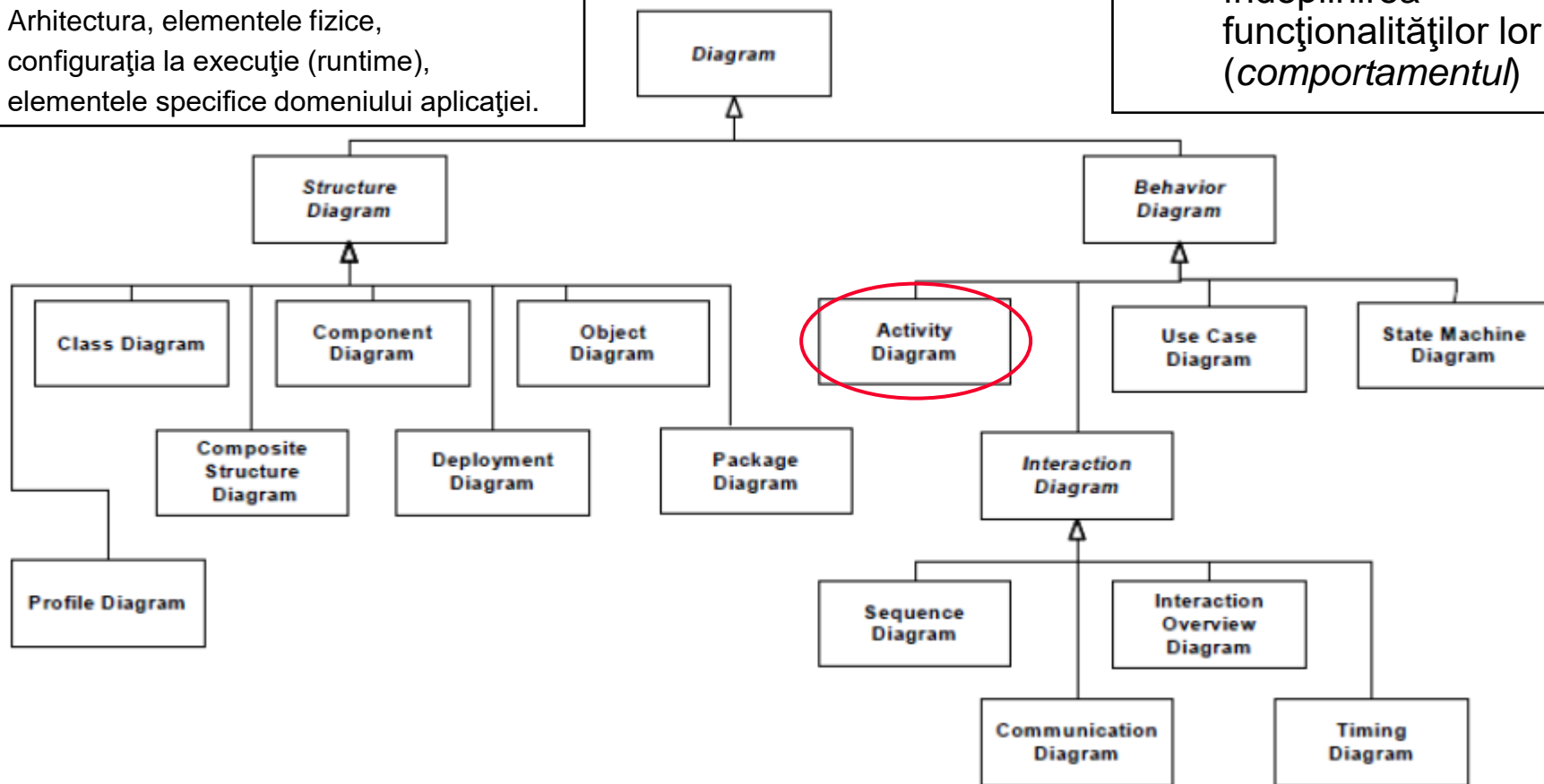


DIAGRAMA DE ACTIVITATE

Reprezentarea fluxului logic de execuție al unui **proces** (fluxul de control și fluxul datelor).

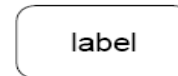
Activitate = unitate de comportament (de nivel înalt)

Elemente componente: *noduri* și *conectori* (arce).

Tipuri de noduri:

- **acțiune** = unitatea *atomică* de comportament executată în contextul unei activități.
- **control** (definirea fluxului acțiunilor)
- **obiect** (transfer de informații între acțiuni)

Action Node:



Control Nodes:



Decision
and
Merge



Fork
and
Join



Initial
node



Final
nodes

Object Nodes:

(used in several ways)

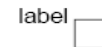
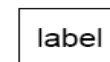
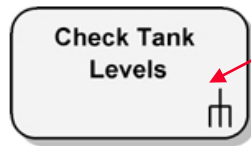


DIAGRAMA DE ACTIVITATE

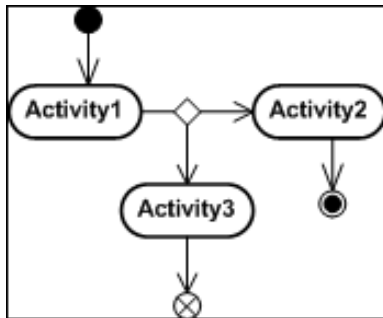
Exemple

NODURI ACȚIUNE

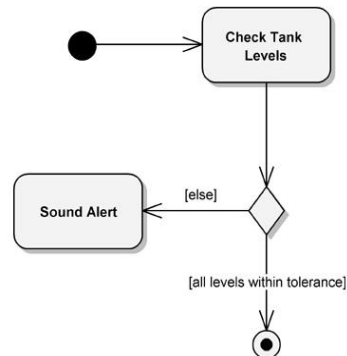
Exemplu: Acțiune predefinită de tip `callBehavior`



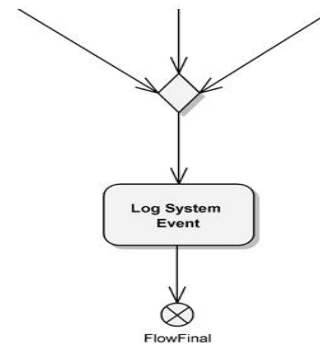
NODURI DE CONTROL



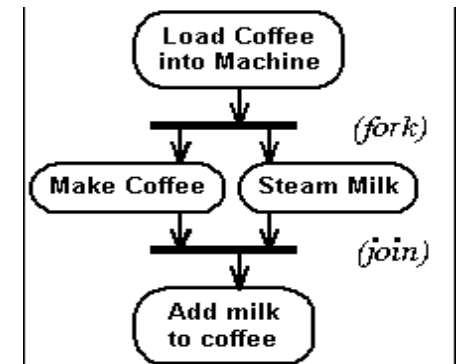
Inițial și finale



decizie



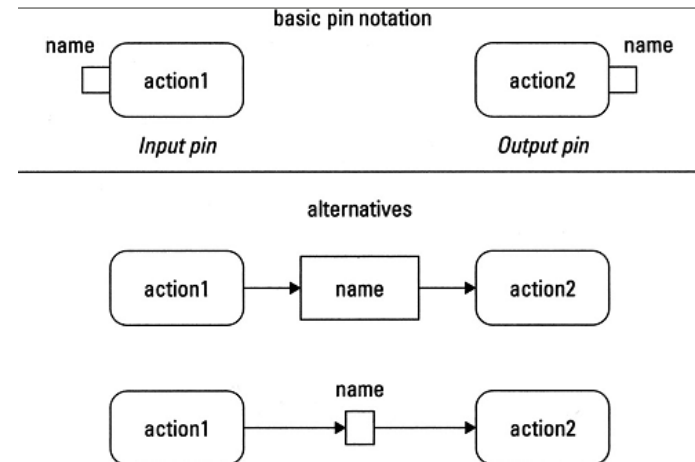
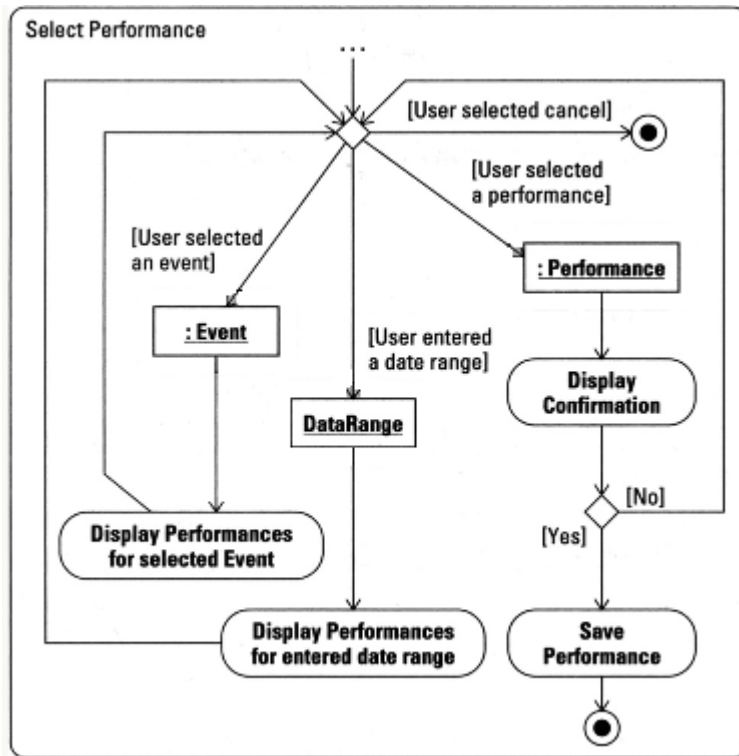
grupare



"fork" și "join"

DIAGRAMA DE ACTIVITATE

NODURI OBIECT



Exemplu:

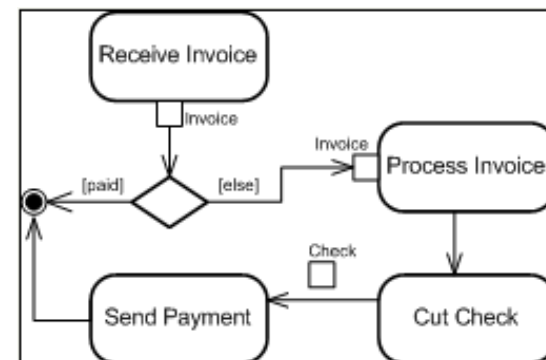
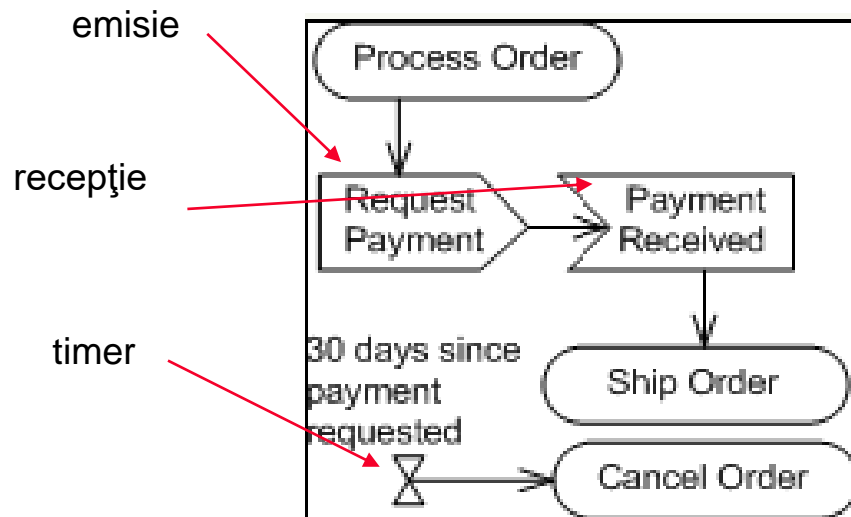


DIAGRAMA DE ACTIVITATE

MODELARE SEMNALE



MODELARE EXCEPȚIE

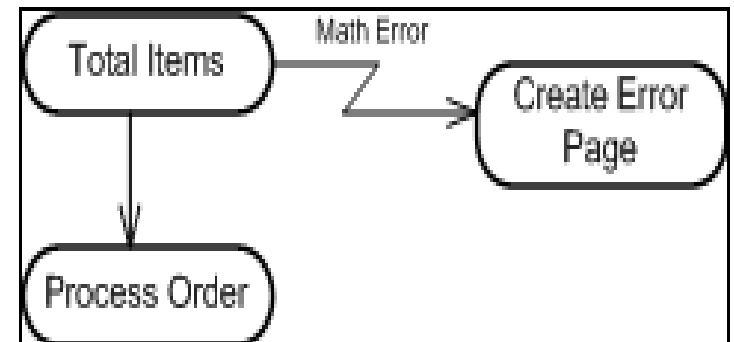


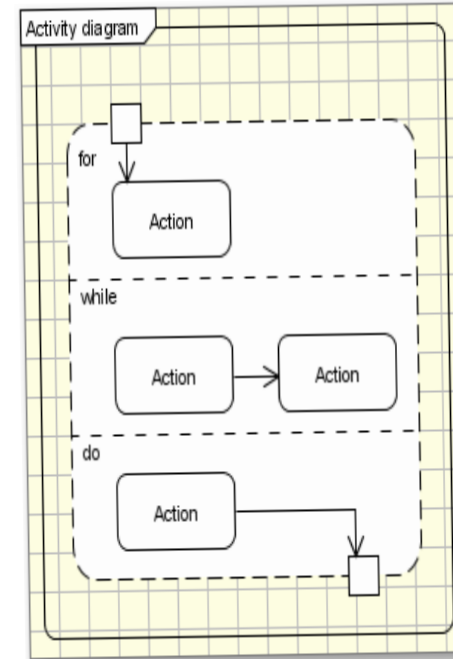
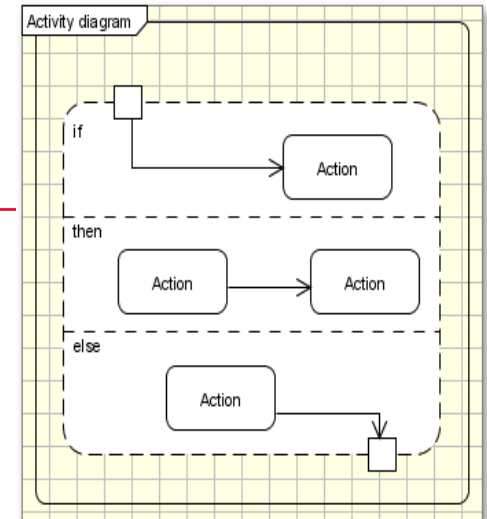
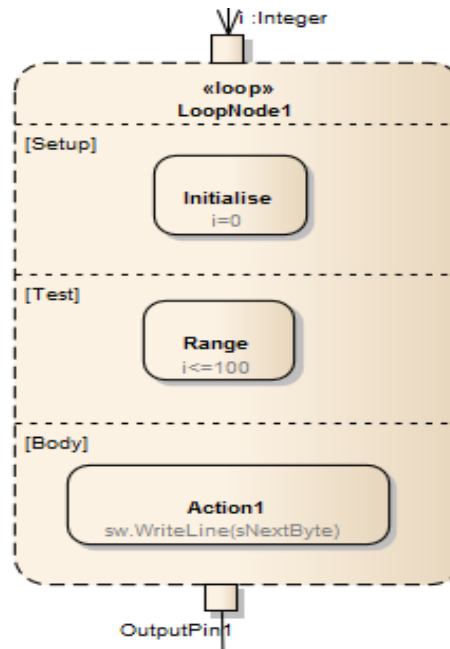
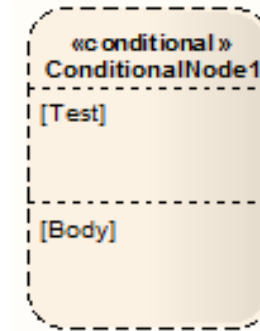
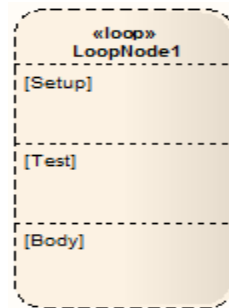
DIAGRAMA DE ACTIVITATE

ACTIVITĂȚI STRUCTURATE

Conditional node = „StructuredActivityNode that chooses one among some number of alternative collections of ExecutableNodes to execute.” - OMG UML 2.5 Specification p. 476.

Loop node = “StructuredActivityNode that represents an iterative loop.

A LoopNode consists of a *setupPart*, a *test* and a *bodyPart*.” - OMG UML 2.5 Specification p. 477



Detalii la : http://www.jot.fm/issues/issue_2005_05/column4/

Exemple preluate de la :

http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/activitydiagram.html

http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/standard_uml_models/loop_and_conditional_nodes2.html

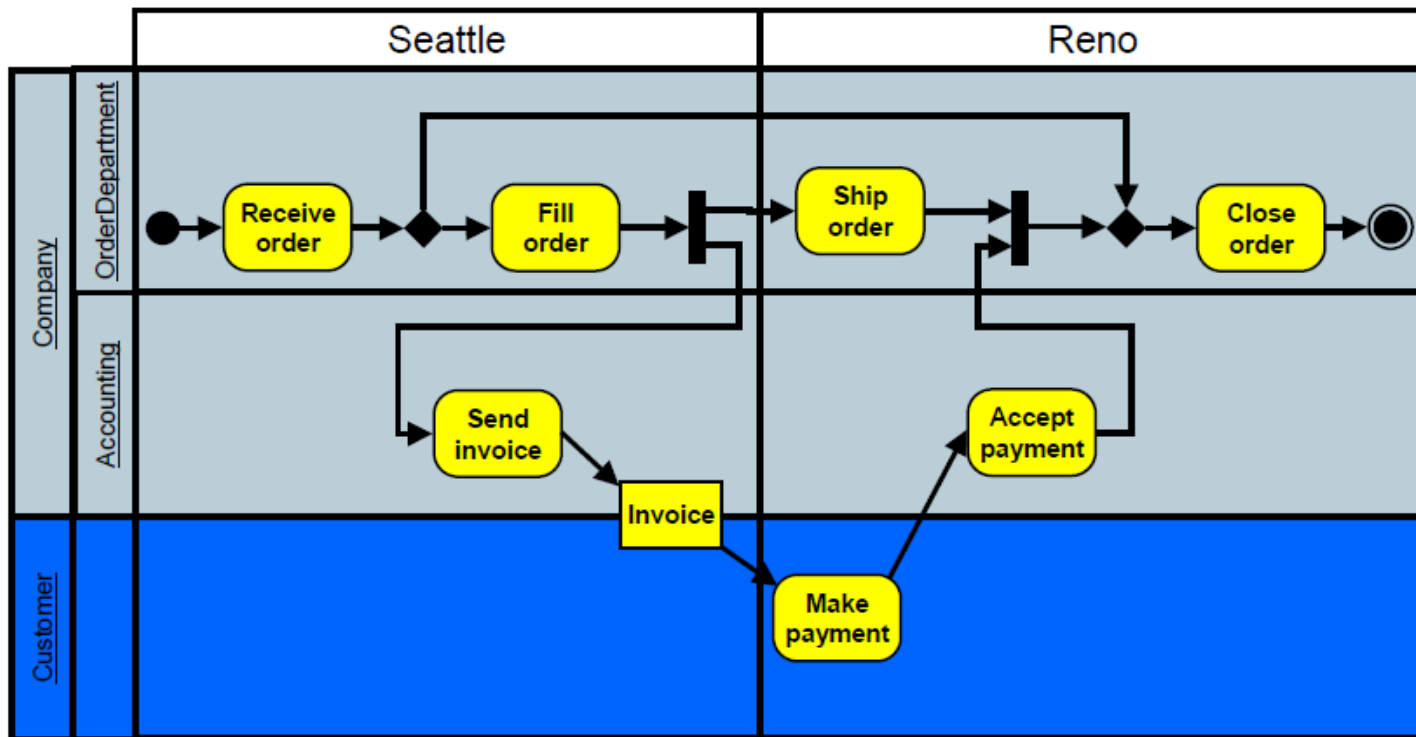
DIAGRAMA DE ACTIVITATE

PARTIȚII (swimlane diagram)

Scop: evidențiere responsabilități.

Diagrama poate avea 1 sau 2 dimensiuni.

Se pot reprezenta sub-partiții



PLAN CURS

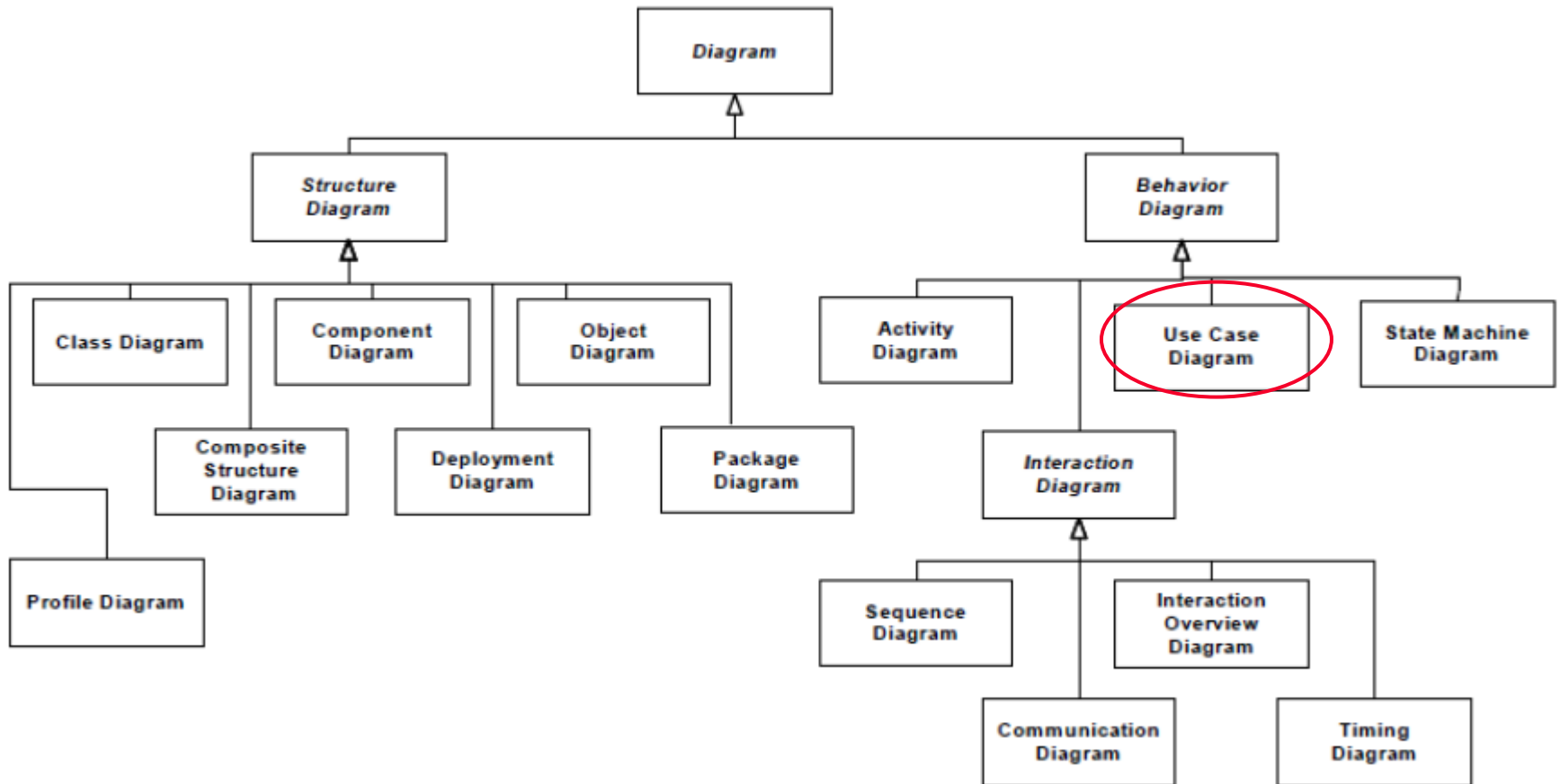


DIAGRAMA DE CAZURI DE UTILIZARE

Ilustrează **contextul** și **funcționalitatea** oferită de sistem din perspectiva utilizatorilor acestuia.

Actor = entitate ce interacționează cu sistemul (persoană sau alt sistem).

Este văzut prin perspectiva **rolului** pe care îl joacă.

Caz de utilizare = secvență de tranzacții corelate realizate de actor și sistem în dialog; curs complet de evenimente petrecute în sistem văzut din perspectiva utilizatorului.

Relație de **asociere**

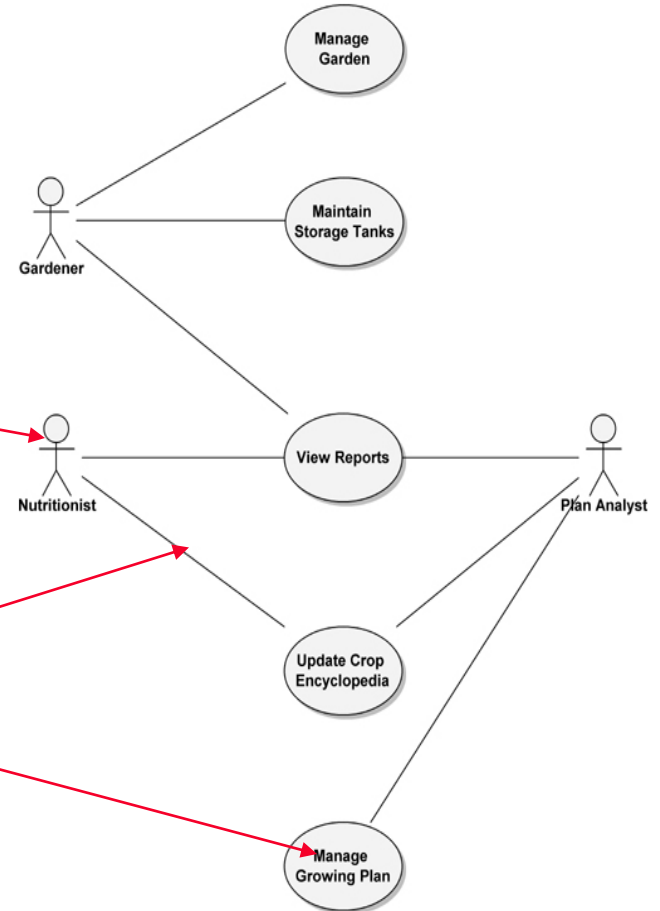


DIAGRAMA DE CAZURI DE UTILIZARE

Tipuri de relații:

Asociere – între actor și cazul de utilizare pe care îl inițiază

Incluziune – incorporarea necondiționată a unui caz de utilizare

Extensie – augmentarea condiționată cu un alt caz de utilizare

Generalizare/Specializare – relație de moștenire între actori sau între cazuri de utilizare.

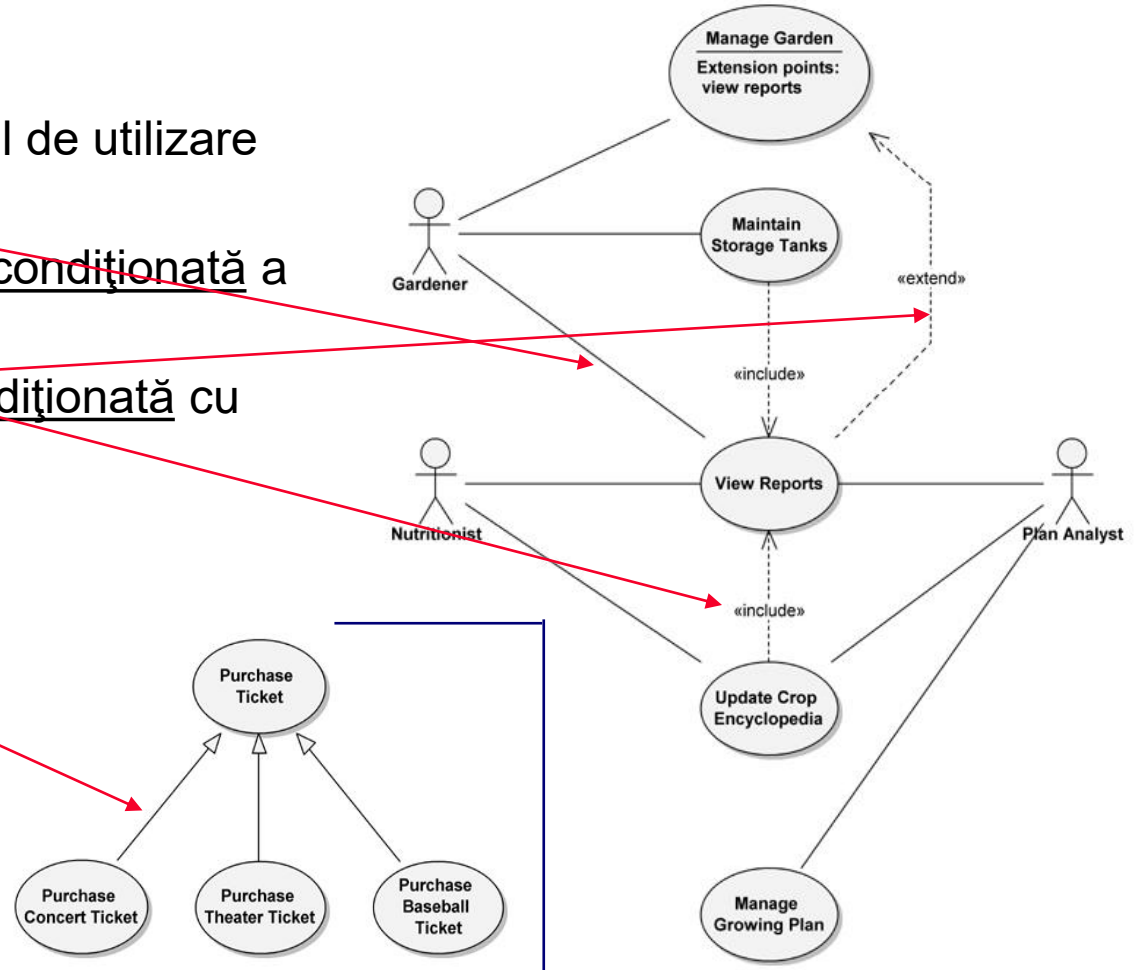


DIAGRAMA DE CAZURI DE UTILIZARE

Abordarea bazată pe cazuri de utilizare: **resurse**

1. *Diagrama* cazurilor de utilizare
2. *Descrierea narativă* a cazurilor de utilizare
3. *Scenariile* cazurilor de utilizare (reprezentate cu diagrame de activitate și/sau diagrame de secvențe la nivel de sistem)

Un caz de utilizare poate avea mai multe scenarii de realizare.

Scenariu într-un caz de utilizare = descrierea unei *singure căi* în executarea unui caz de utilizare.

Scenariile pot forma baza unui *plan de testare* al cazului de utilizare.

Etapele dezvoltării diagramei de cazuri de utilizare:

1. Definirea contextului sistemului
 - 1.1 Identificarea actorilor și responsabilităților lor
 - 1.2 Identificarea cazurilor de utilizare (comportamentele sistemului în termeni de obiective și/sau rezultate ce trebuie produse)
2. Evaluarea actorilor și a cazurilor de utilizare pentru identificarea oportunităților de rafinare: divizarea sau gruparea definițiilor.
3. Evaluarea cazurilor de utilizare pentru identificarea relațiilor de incluziune și de extindere.
4. Evaluarea actorilor și a cazurilor de utilizare pentru identificarea oportunităților de generalizare (detectarea proprietăților comune).

PLAN CURS

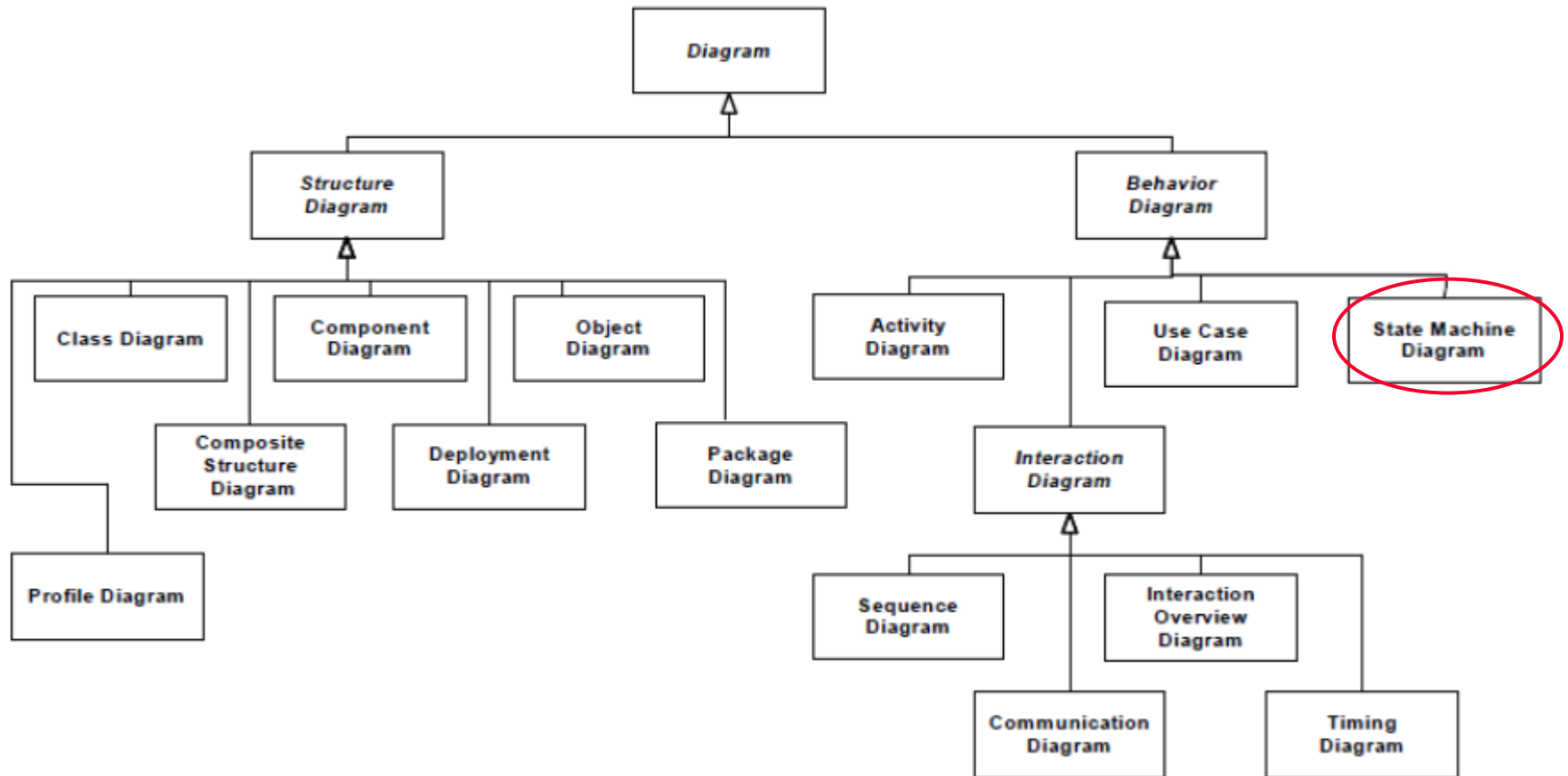


DIAGRAMA DE STĂRI

Exprimare **comportament** prin:

- tranzițiile de la o stare la alta în cadrul unui set de stări
- evenimentele ce declanșează tranzițiile
- posibile acțiuni corelate cu tranziții

Exprimă comportamentul pe diferite nivele de abstractizare (ex. pentru un obiect, subsistem sau sistem).

Stare obiect = rezultatul cumulativ al comportamentului obiectului, definită de valorile proprietăților sale (atribute și relații).

În cadrul unei stări, un obiect poate:

- executa o activitate
- aștepta un eveniment
- îndeplini o condiție.

ELEMENTE ESENȚIALE:

- stare
- tranziție
- eveniment

DIAGRAMA DE STĂRI

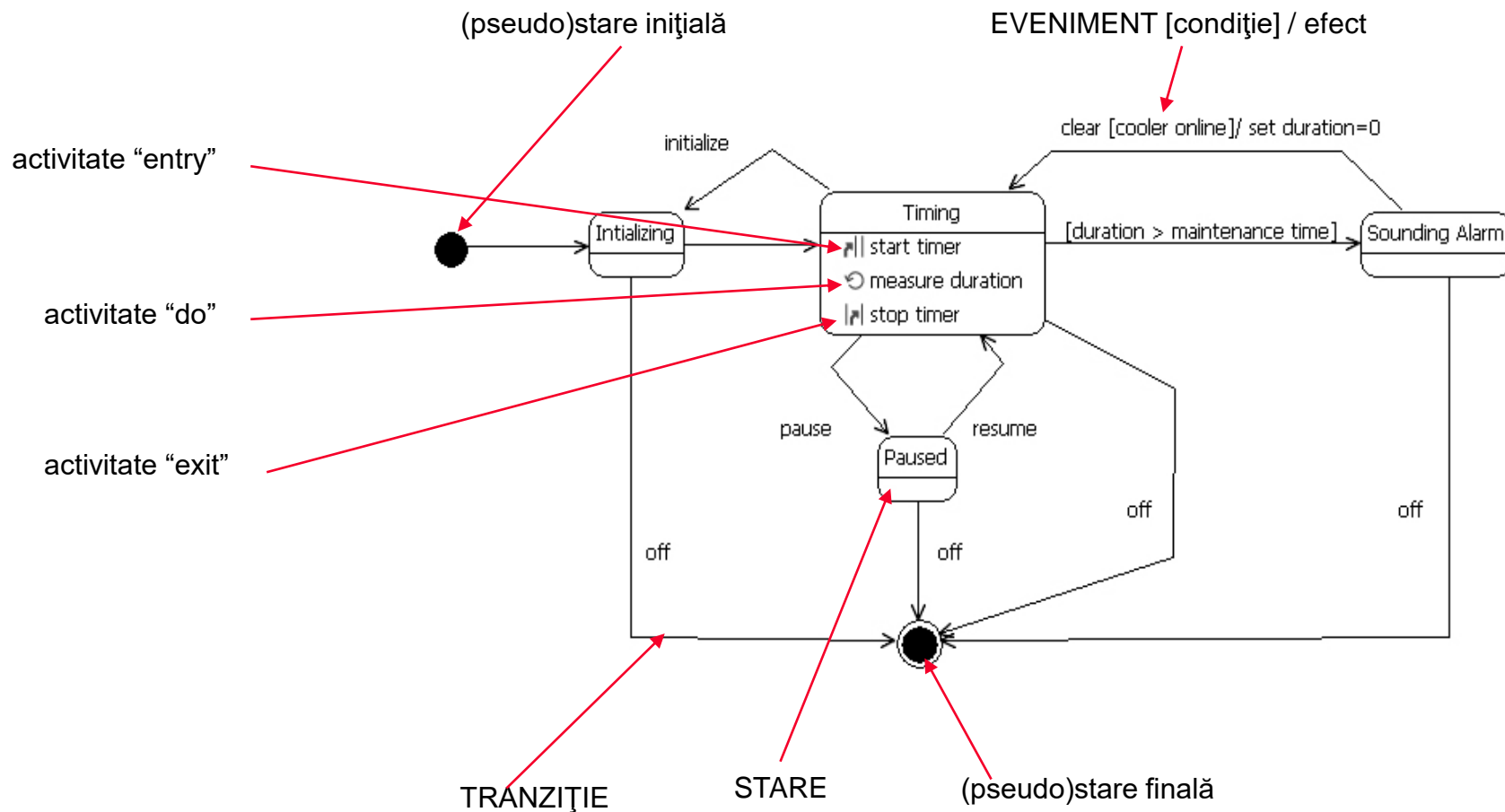


DIAGRAMA DE STĂRI

STĂRI COMPUSE ȘI STĂRI INCLUSE

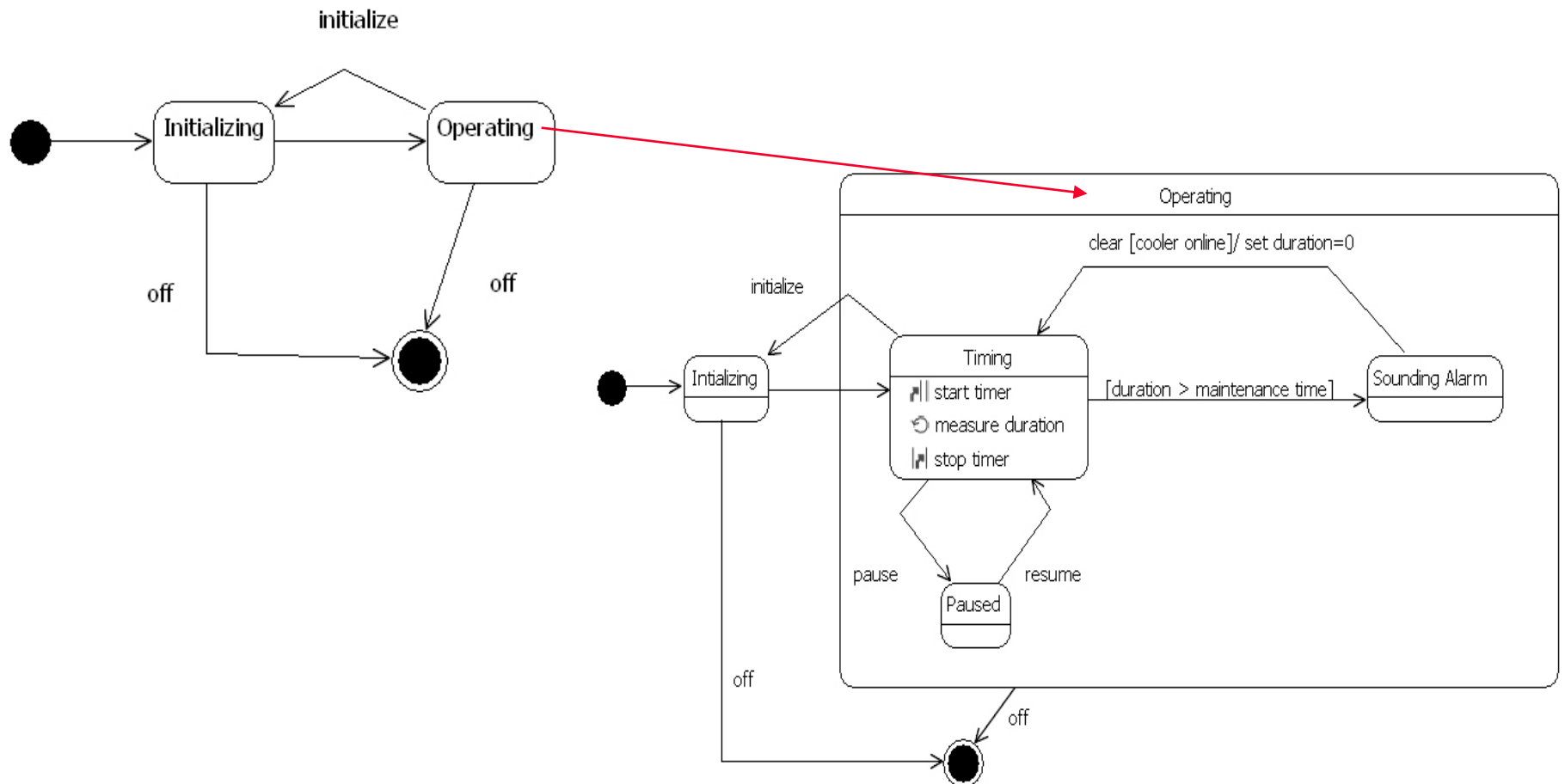
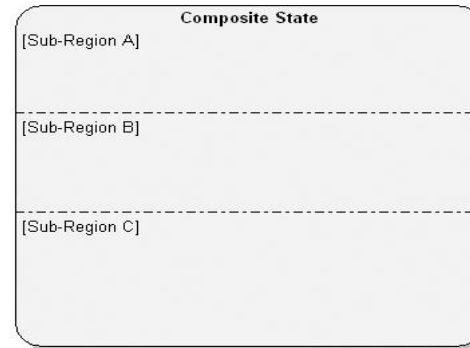


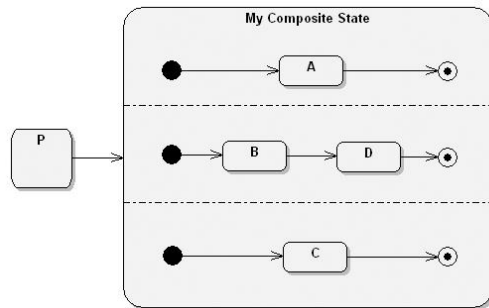
DIAGRAMA DE STĂRI

CONCURENȚĂ ȘI CONTROL

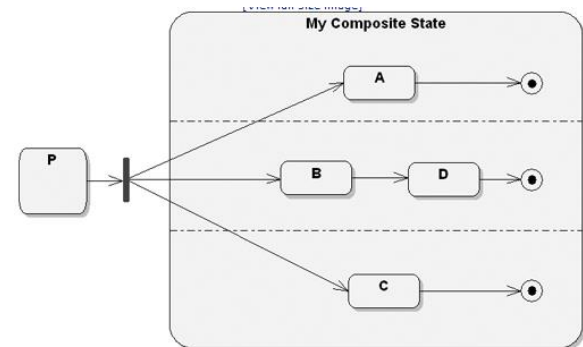


Stare compusă :
reprezentare generală

Variante de tranziție către o stare compusă



Activarea tuturor submașinilor concurente,
din stările inițiale

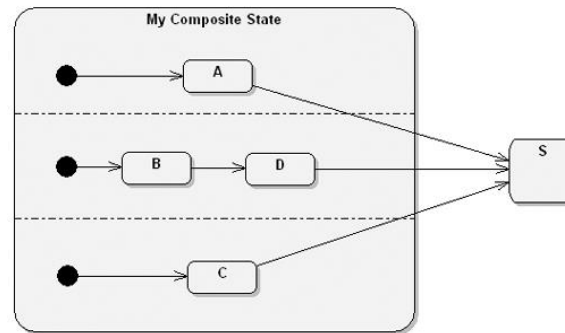
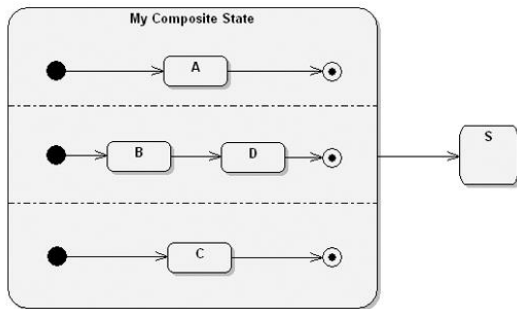


Precizarea substărilor către care se fac
tranziții concurente.

DIAGRAMA DE STĂRI

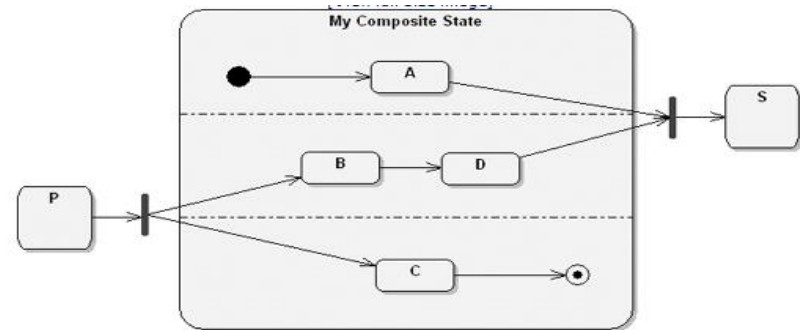
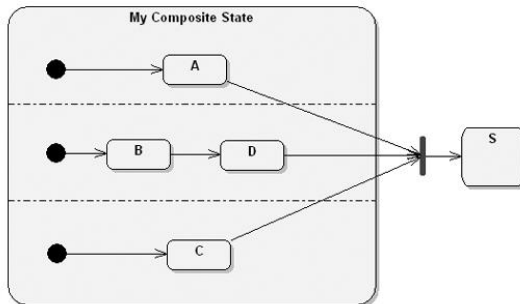
CONCURENȚĂ ȘI CONTROL

Variante de ieșire dintr-o stare compusă



Oricare din tranzițiile de ieșire va forța realizarea celorlalte și ieșirea din starea compusă.

Atunci când toate submașinile ajung în stare finală.



Activarea din starea inițială și trecerea în starea finală sunt forțate de tranziții ale unor submașini concurente.

PLAN CURS

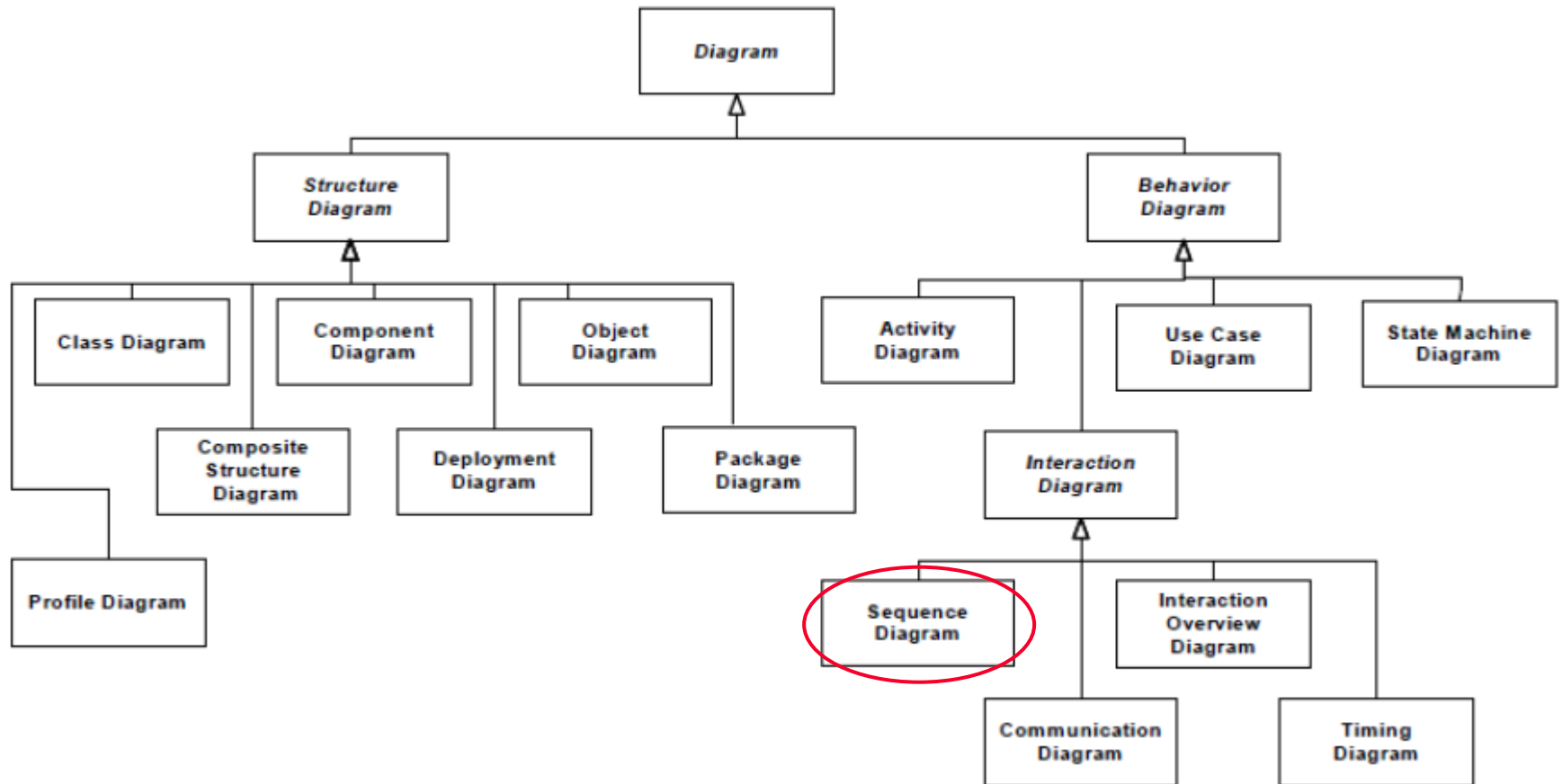


DIAGRAMA DE SECVENȚE

Perspectivă orientată în timp asupra ***transferurilor de mesaje*** dintre obiecte.

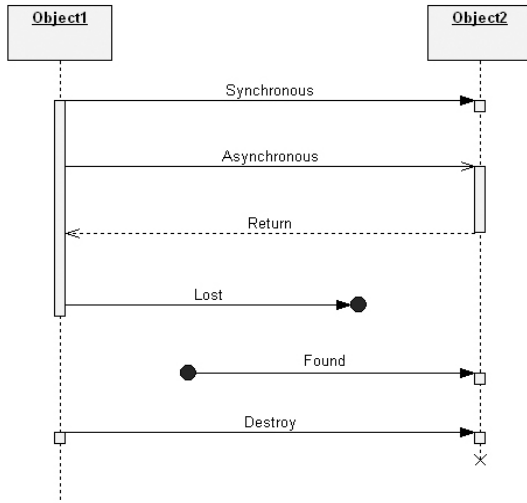
Se poate reprezenta pe diferite *nivele de abstractizare*:

- interacțiuni între subsisteme (↑)
- interacțiunii între instanțe pentru o singură operație sau activitate (↓)

Nu este evidențiat timpul absolut ci *ordinea mesajelor*.

DIAGRAMA DE SECVENȚE

TIPURI DE MESAJE



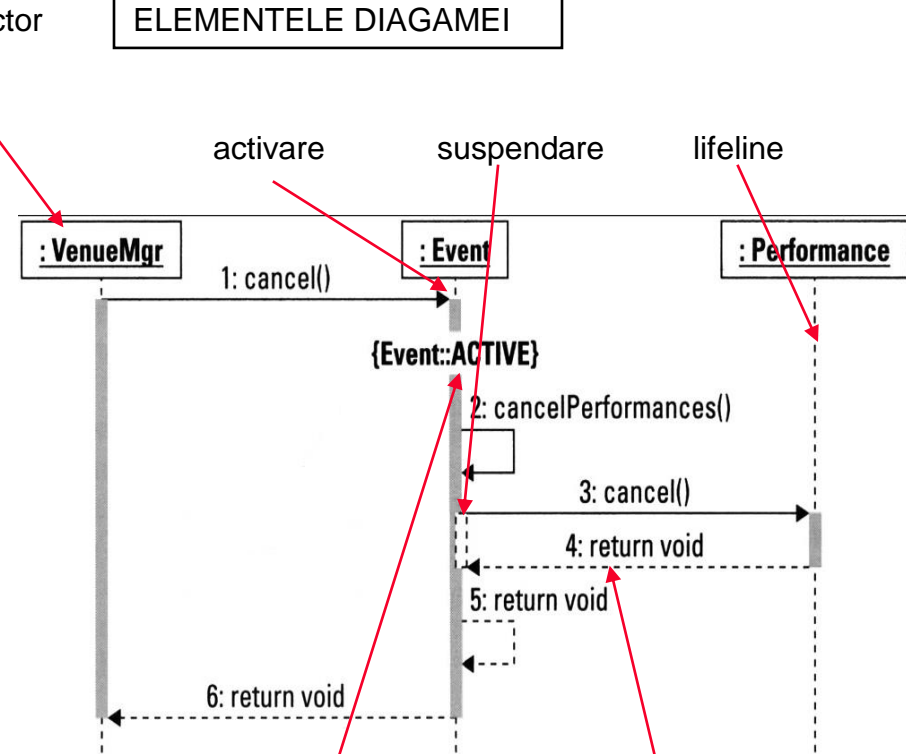
Sintaxă mesaj:

```
[attribute =] signal or-operation-name [ (
arguments ) ] [: return-value] | '*'
```

```
arguments ::= argument [ , arguments]
```

```
argument ::= [parameter-name=] argument-value
| attribute= out-parameter-name [:argument-value]
```

ELEMENTELE DIAGAMEI



expresie definire invariant stare
(precondiție pentru următoarele
transferuri de mesaje)

mesaj

DIAGRAMA DE SECVENȚE

DIAGRAME COMPLEXE

FRAME = context portabil pentru o diagramă

Poate fi reutilizat (inclus) în alt frame
sau în altă diagramă

[<kind>]<name>[<parameters>]

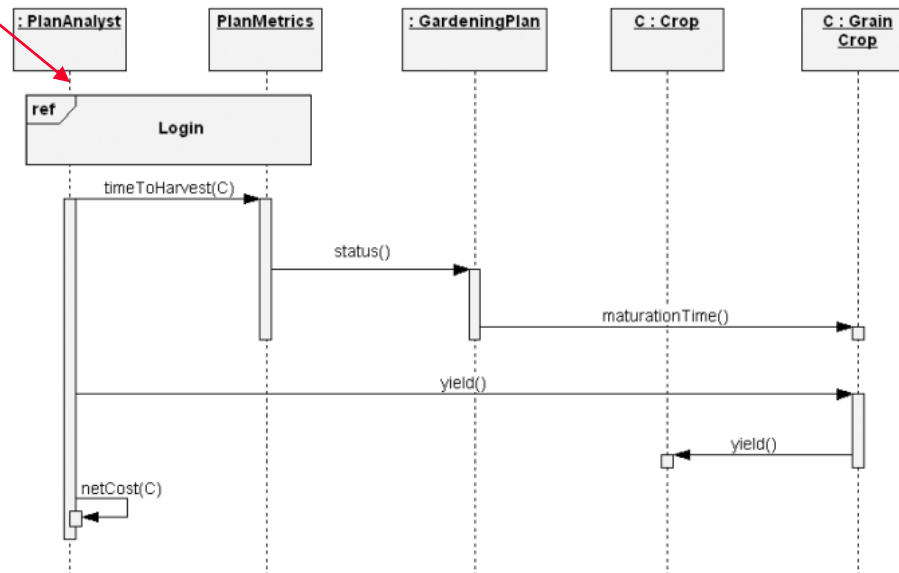
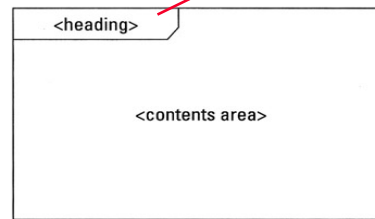
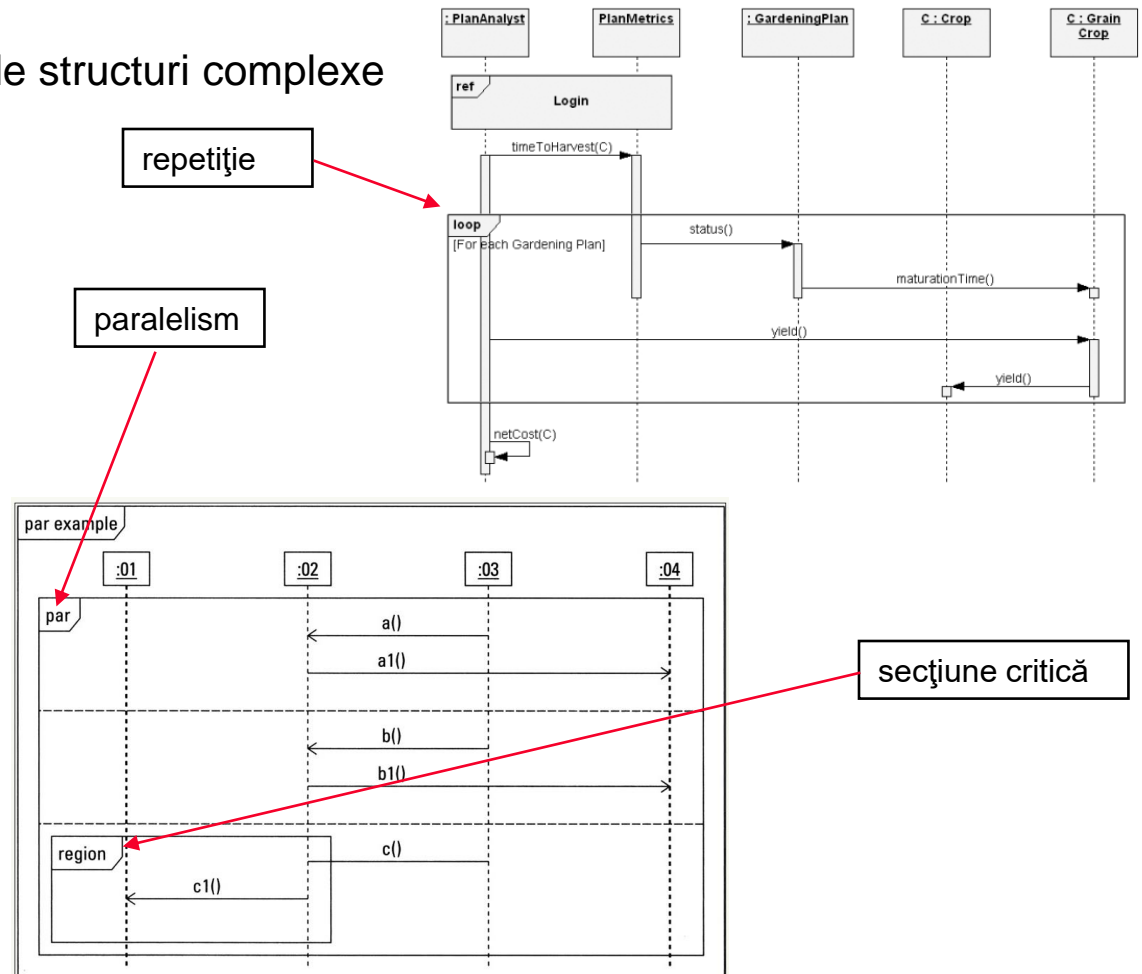
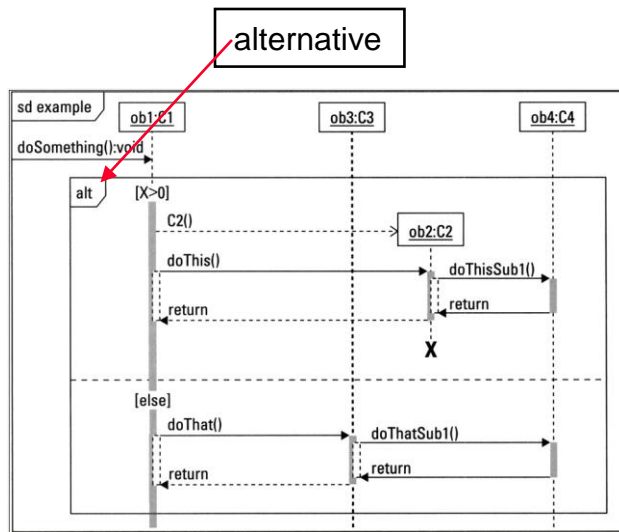


DIAGRAMA DE SECVENȚE

DIAGrame COMPLEXE

Combinare frame-uri pentru creare de structuri complexe



PLAN CURS

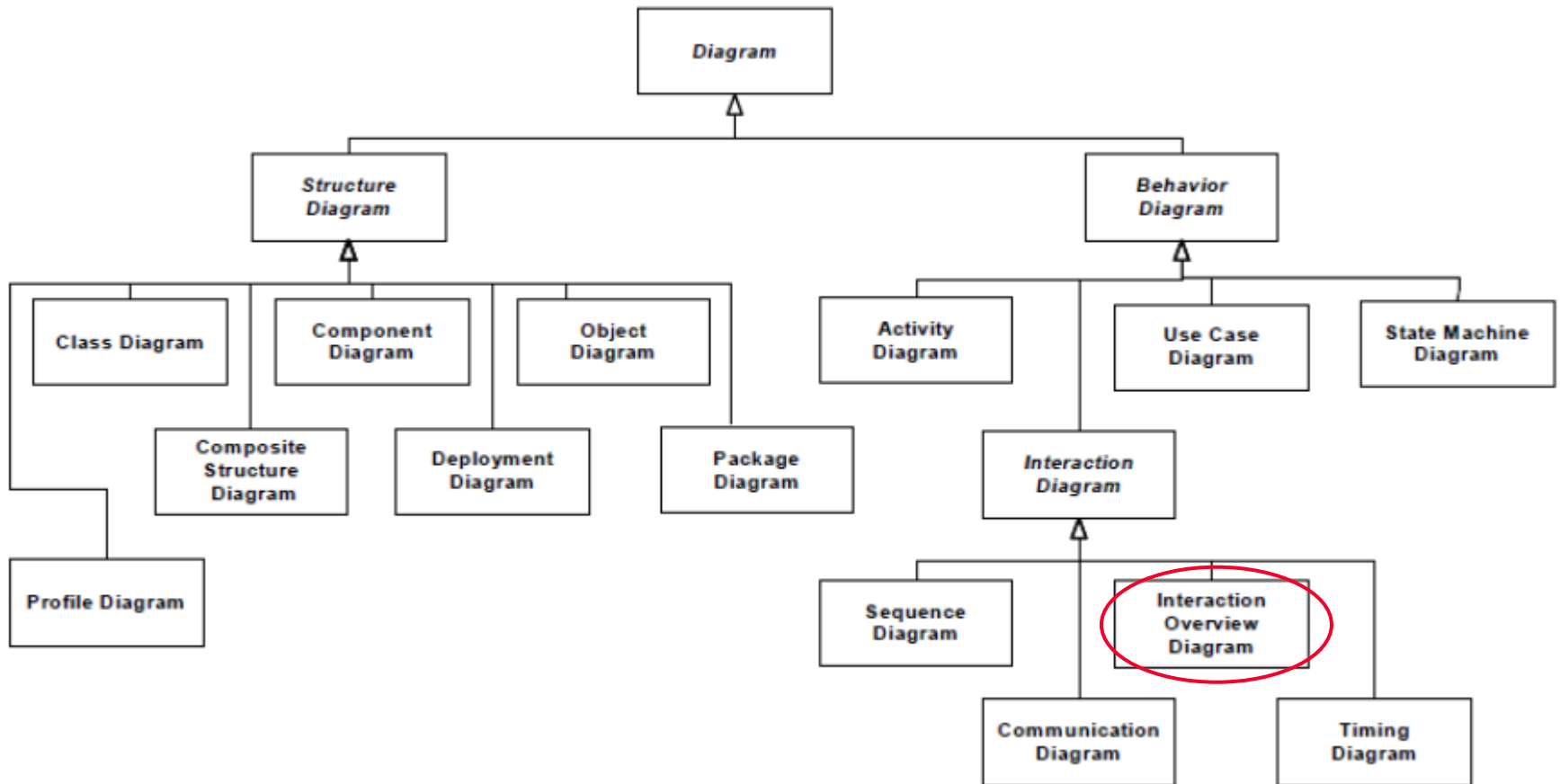


DIAGRAMA DE ANSAMBLU A INTERACȚIUNII

Combină:

- fluxul de control (*proces*)
- specificarea transferurilor de mesaje (*interacțiuni*)



Diagrama de activitate

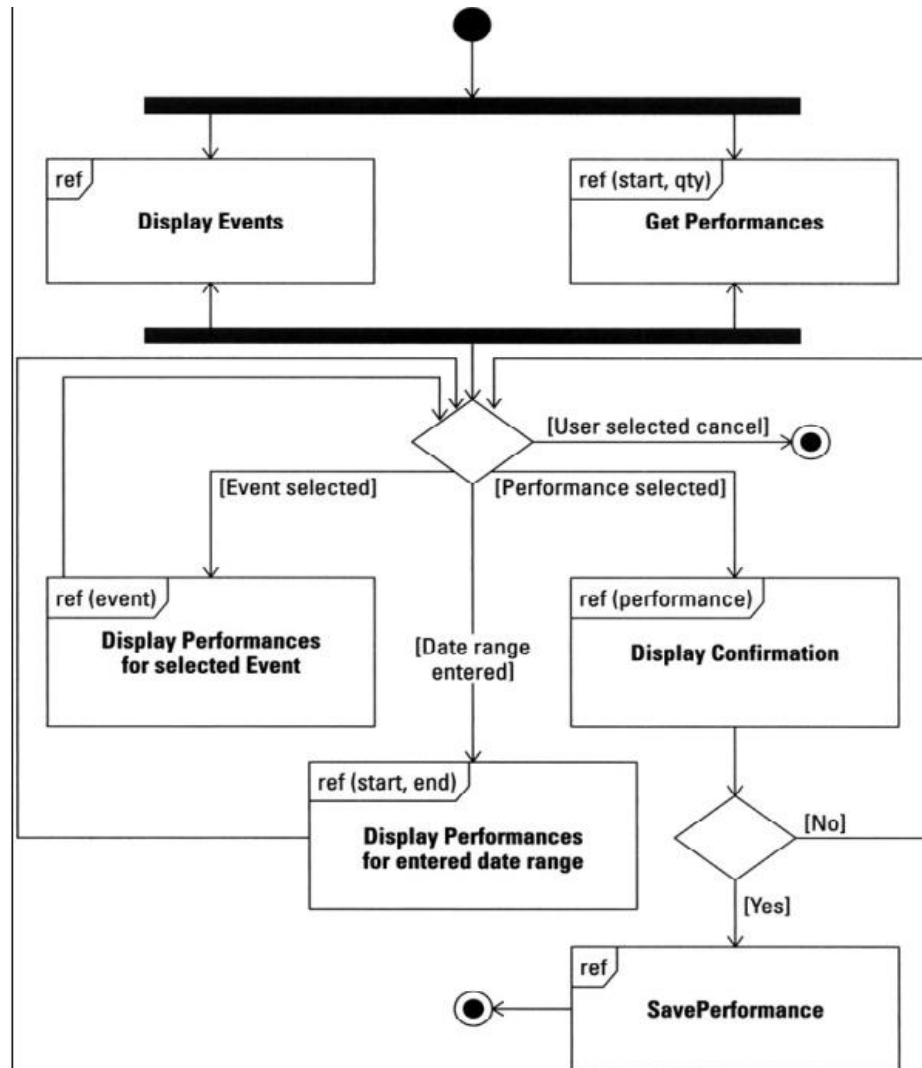
Utilizează activități și fluxuri de obiecte pentru a descrie un proces.



Diagramă de secvențe

Utilizează interacțiuni pentru modelarea fluxului logic al unui singur scenariu (o cale logică într-un caz de utilizare).

DIAGRAMA DE ANSAMBLU A INTERACȚIUNII



PLAN CURS

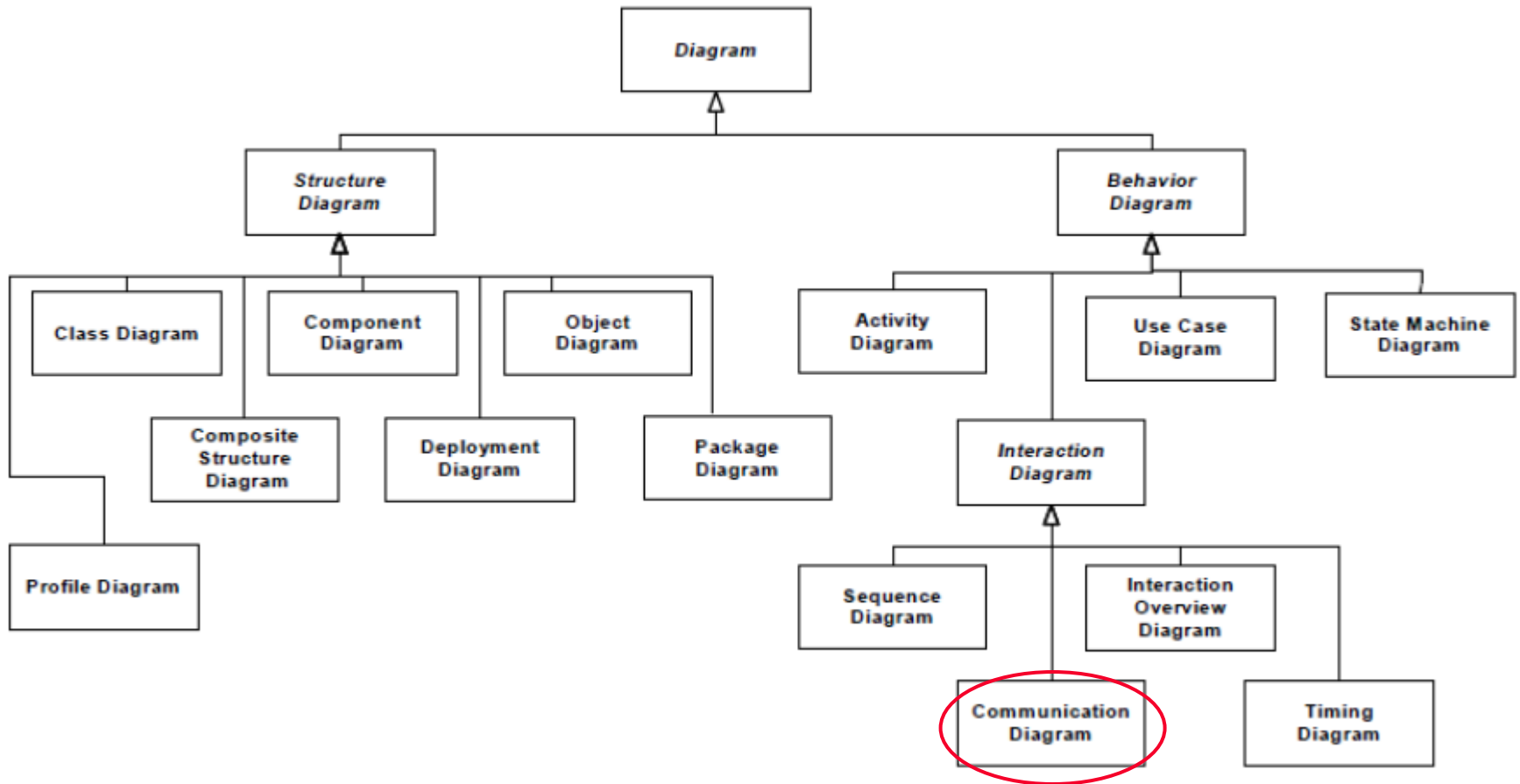
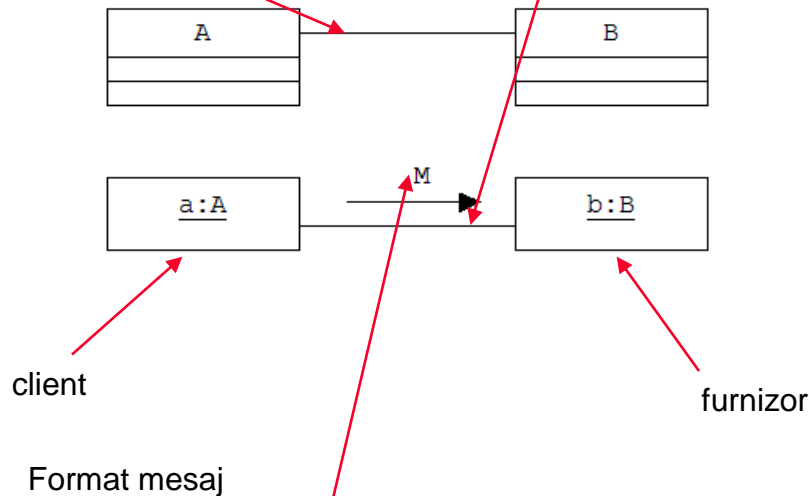


DIAGRAMA DE COMUNICARE

Reprezentare:

- legăturile dintre obiecte
- mesajele transferate în cursul interacțiunii

Relație între clase \Rightarrow cale de comunicare (transfer mesaje) între obiecte.

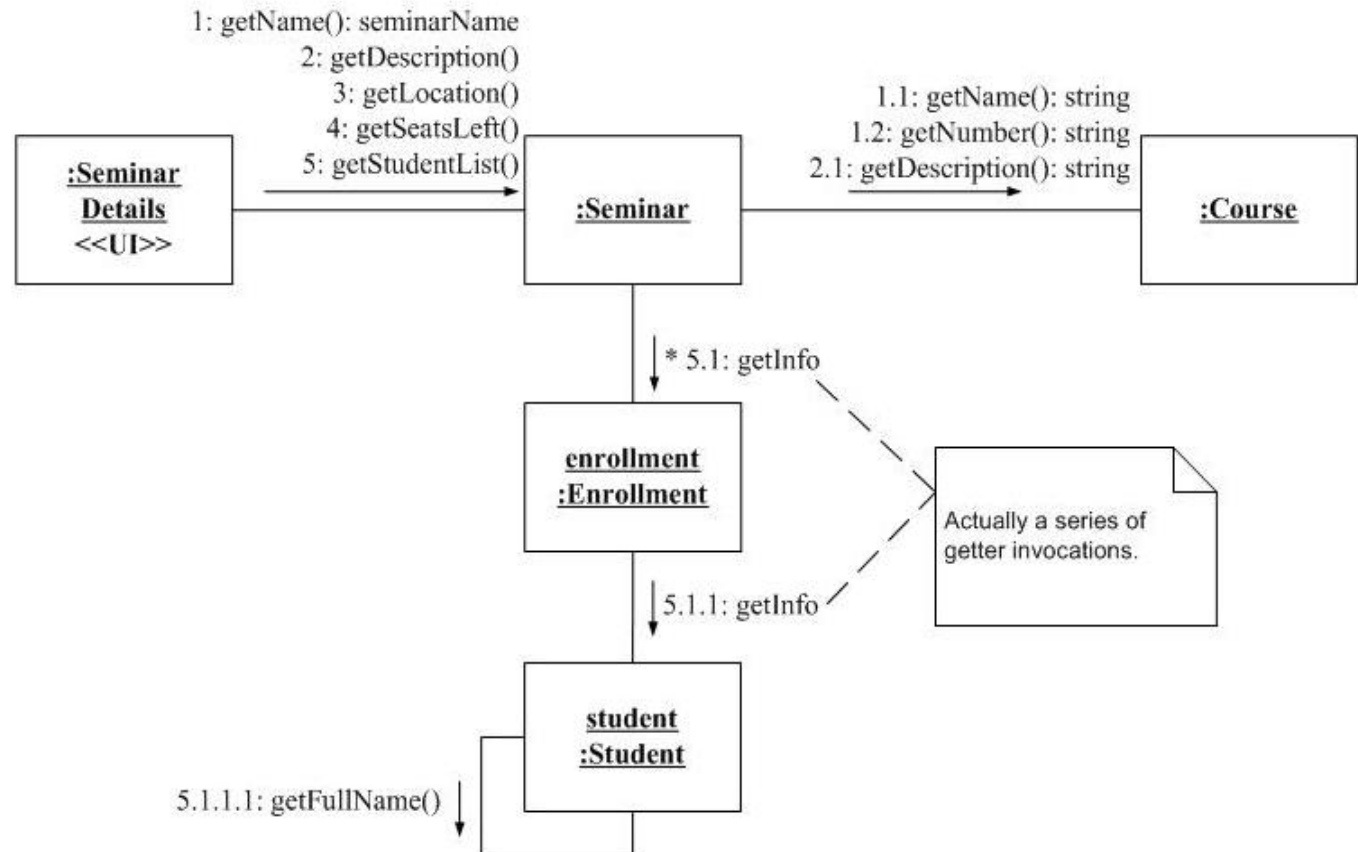


Clasa B (sau o superclasă a sa)
trebuie să conțină declarația lui M.

[NrSecvență:] NumeMetodă(listă parametri) [: ValoareReturn]

DIAGRAMA DE COMUNICARE

Exemplu:



PLAN CURS

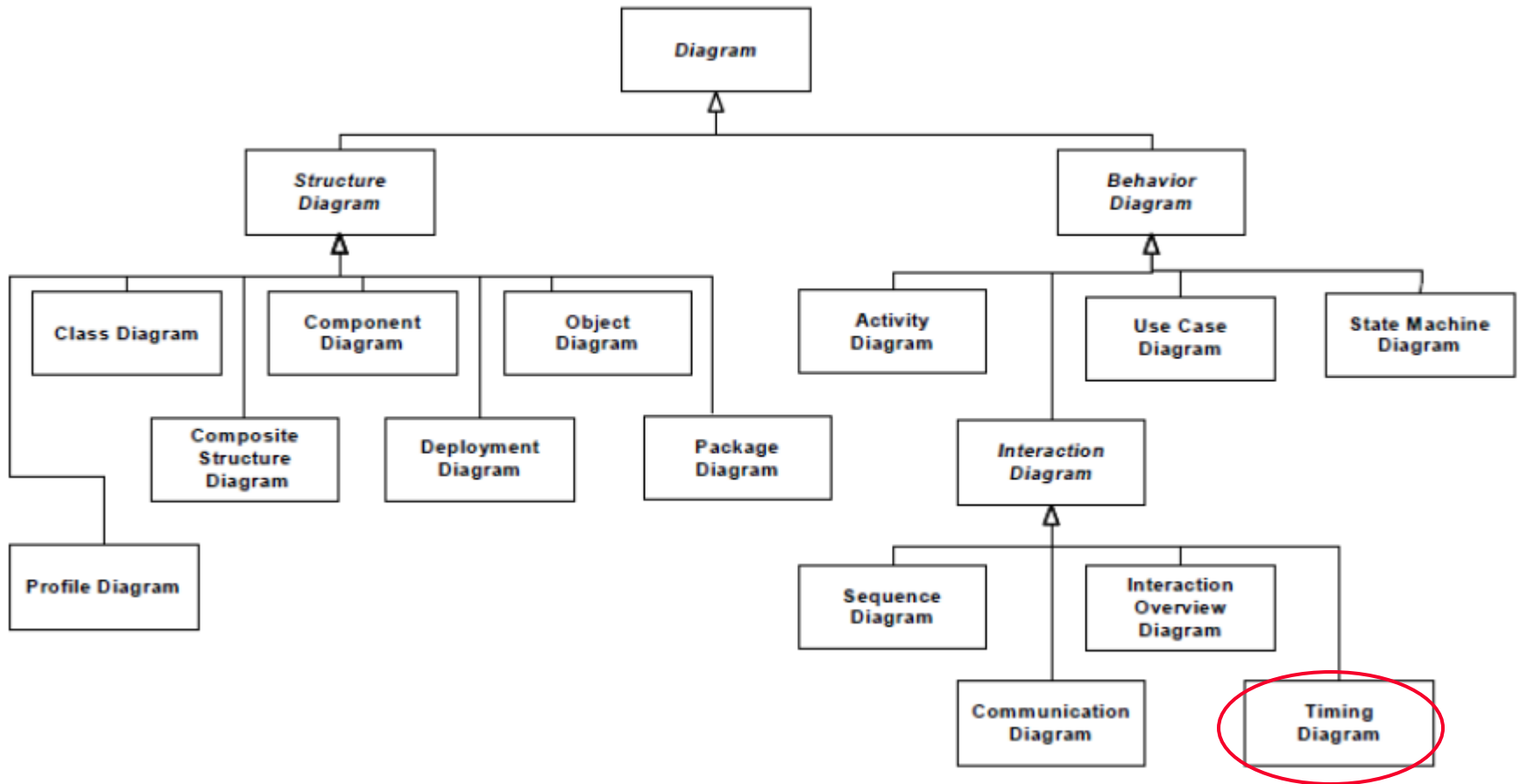
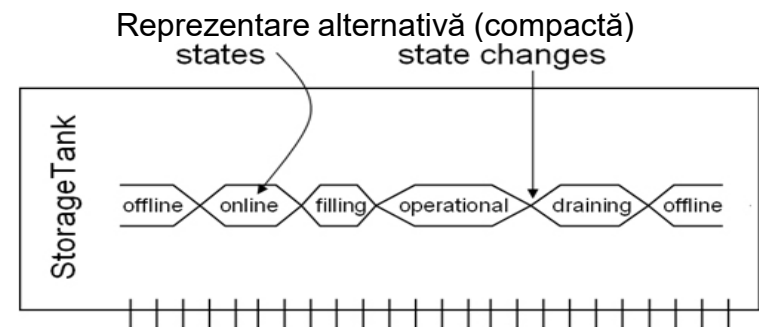
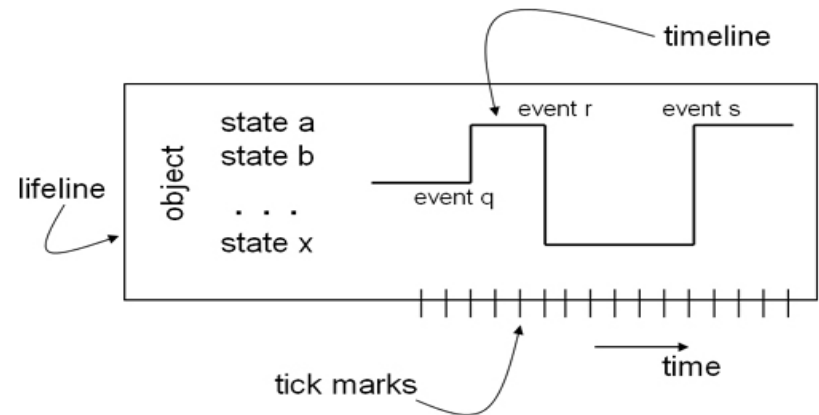
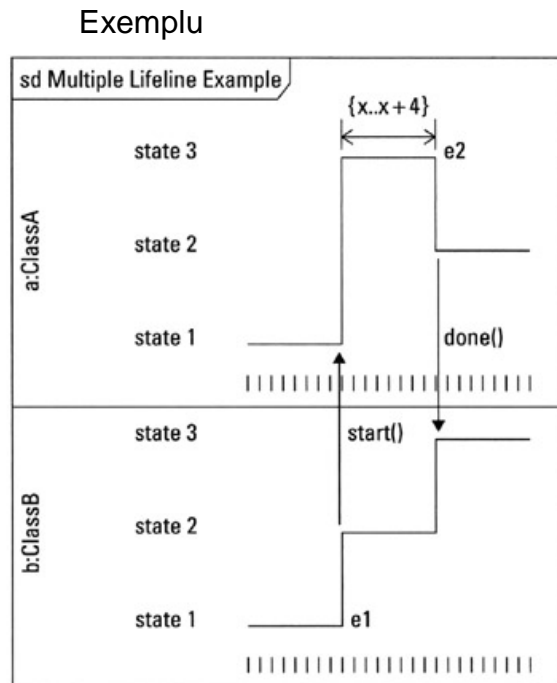


DIAGRAMA DE TIMP

Reprezentarea în timp a schimbărilor de stare și a evenimentelor ce le declanșează.

Diagramă de timp generică



METAMODELARE

UML Metamodel* – specifică sintaxa abstractă a limbajului UML

- Setul *conceptelor de modelare* UML
- *Atributele* conceptelor
- *Relațiile* între concepte
- *Regulile de combinare* a conceptelor pentru construire modele UML
- *Semanticile* fiecărui concept de modelare
- *Reprezentările grafice* ale conceptelor și relațiilor

Exemplu : Specificația UML 2.5.1 - <https://www.omg.org/spec/UML/2.5.1/PDF>

UML Model – model al unei aplicații definit folosind limbajul UML
(diagrame UML)

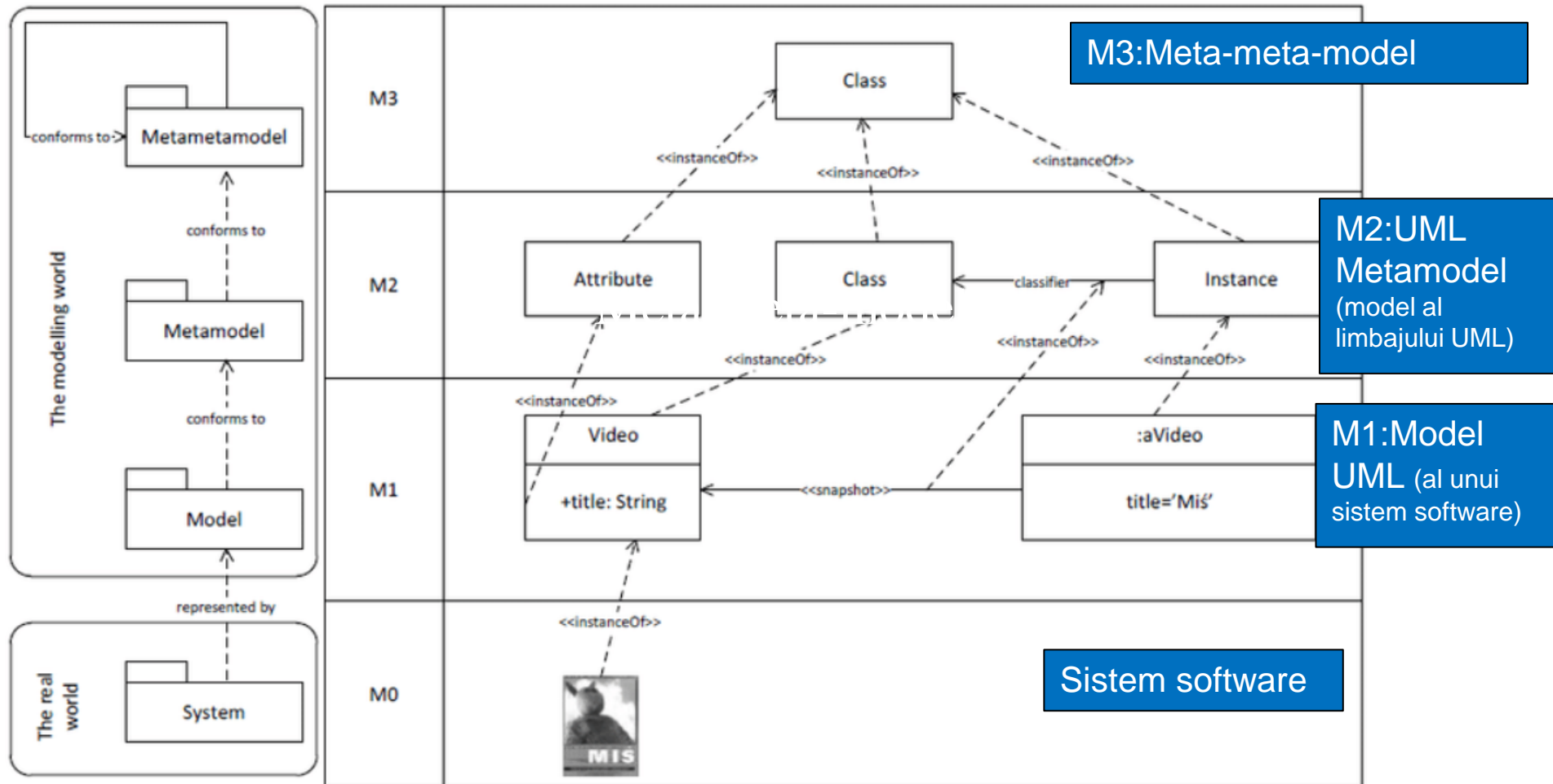
Instanță model – instanțe de concepte și de relații; obiecte din
lumea reală

Exemplu simplu :

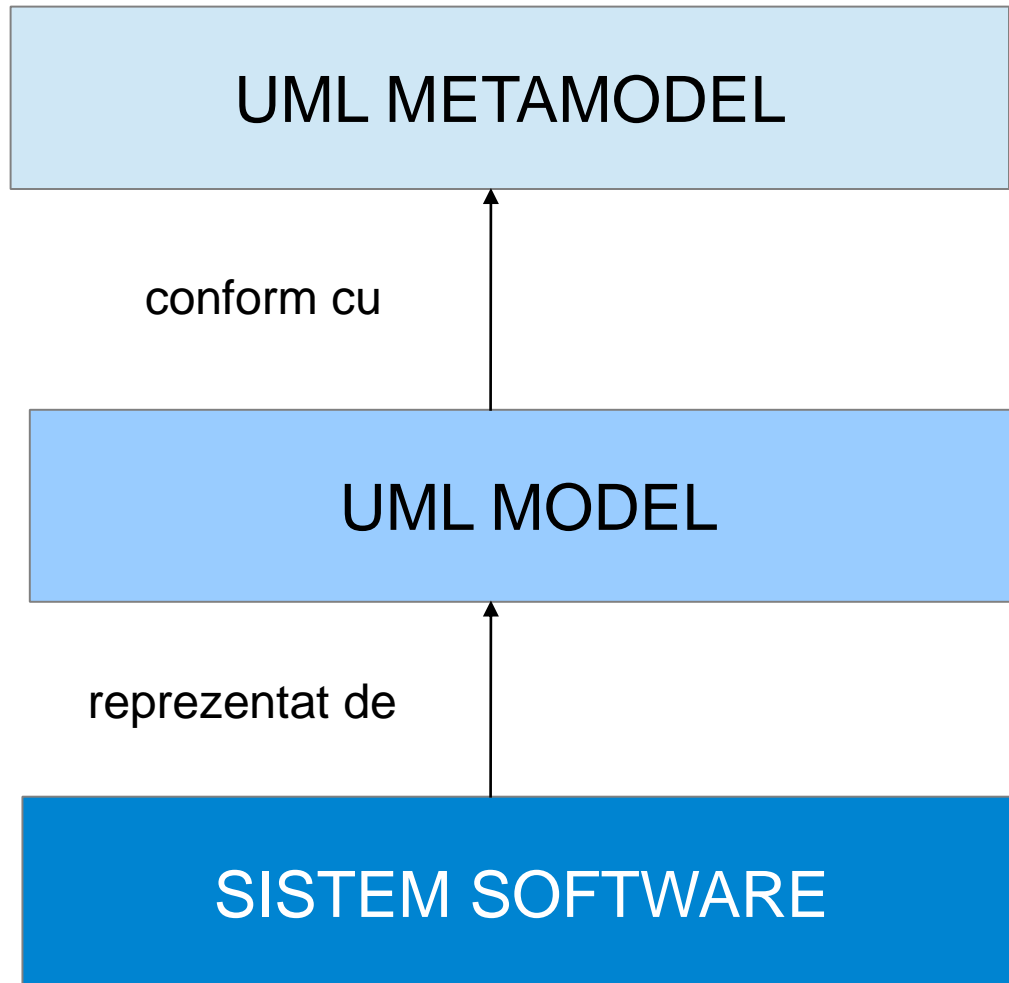
În metamodelul UML – conceptul **class**
În modelul unei aplicații – clasa **Student**
Instanță la execuție – obiectul **Popescu**

* model of a modeling language

METAMODELARE

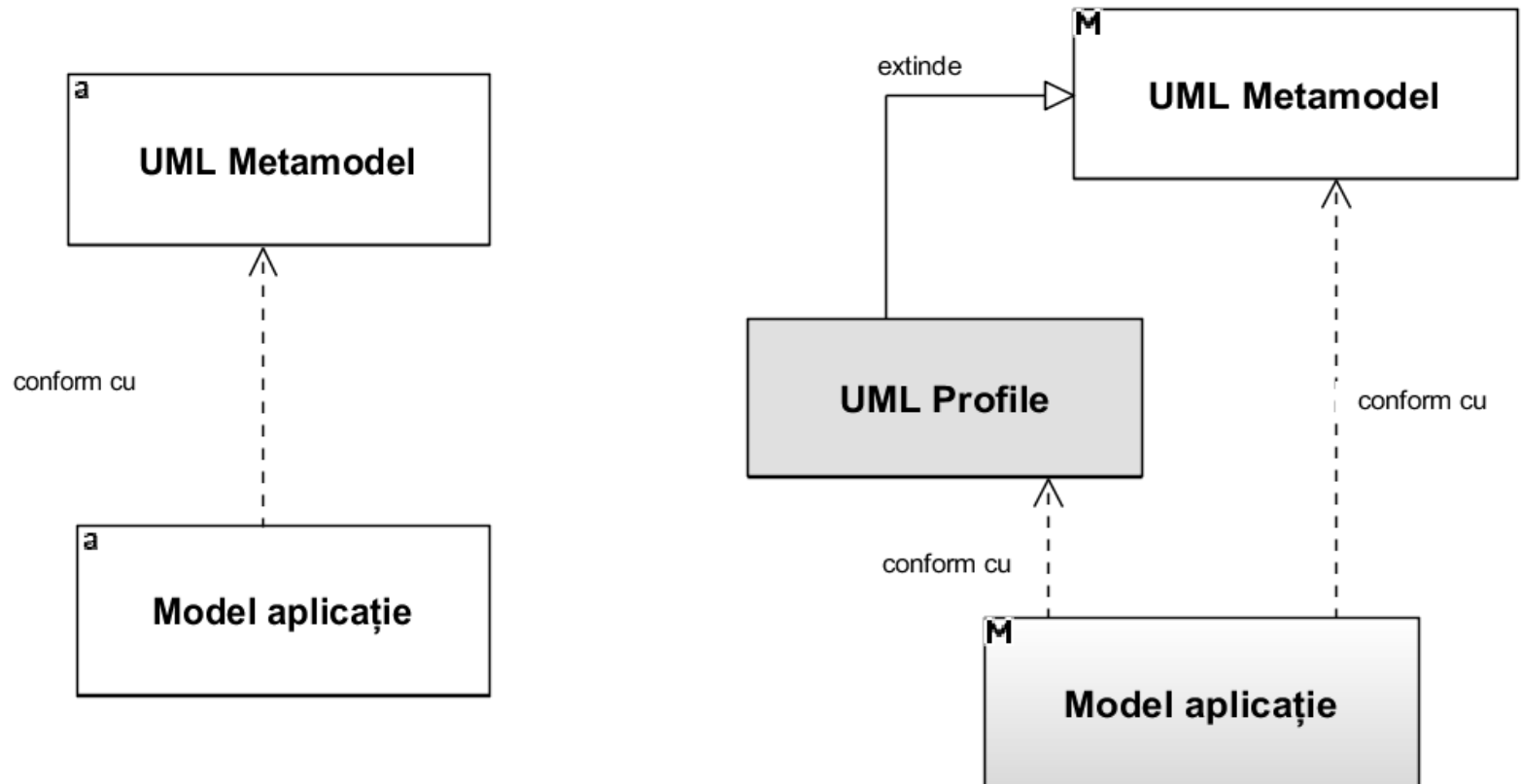


METAMODELARE



PROFIL UML

Profil UML = limbaj de modelare derivat din UML



MECANISME PENTRU EXTINDEREA UML-lui

Stereotipuri - Rafinarea *interpretării semantice* a elementelor modelului.

TAG-uri – abilitatea de a asocia *informații suplimentare* unui element al modelului.

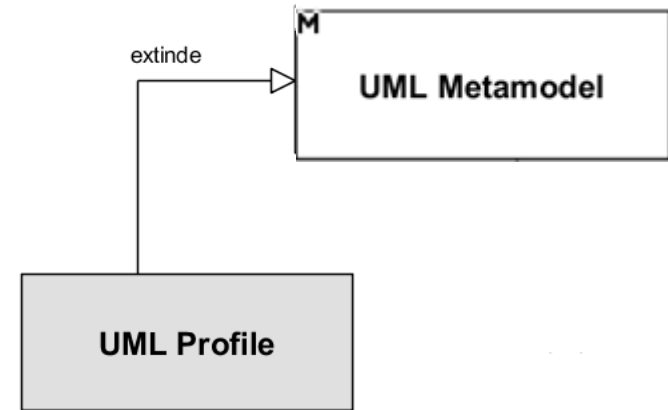
Definește *numele* și *tipul* informației asociate.

TAG VALUE – instanță a unei definiții de TAG, cu o valoare de tipul celei din definiția TAG-lui.

CONSTRÂNGERI – *reguli* aplicate unor elemente ale modelului.
(reprezentate între acolade).

OCL (*Object Constraint Language* - altă specificație OMG) – limbaj pentru definire de constrângeri.

PROFILE UML



Obs. Tipurile de elemente din diagramele UML – definite în metamodelul standardului UML.

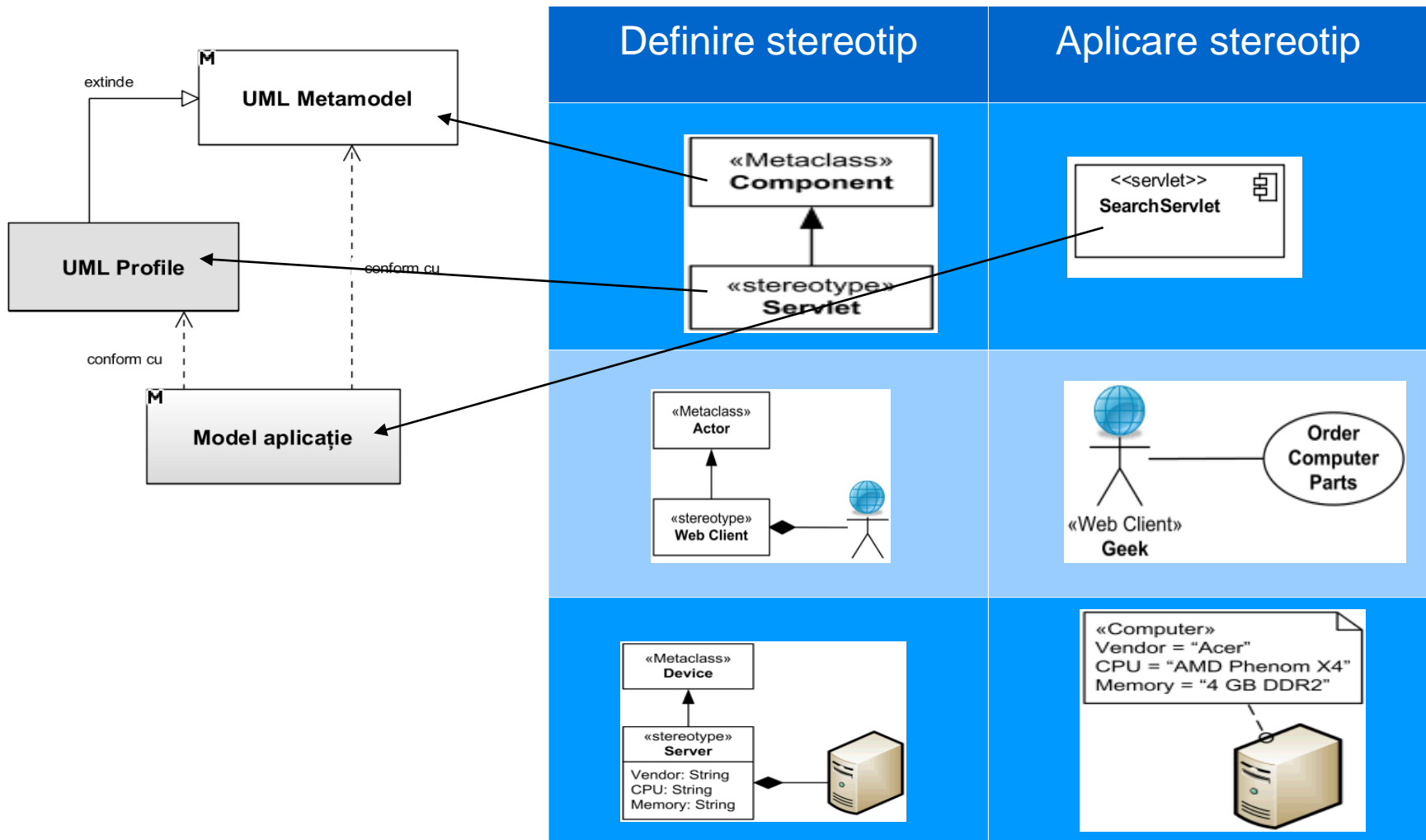
(Exemple de tipuri : class, attribute, association, component, port, use case, actor, etc.)

PROFIL UML – colecție de *stereotipuri*, *definiții de tag-uri* și *constrângeri* utilizată pentru a defini un nou set de semantici pentru un model.

Profil – adaptarea metamodelului UML cu construcții specifice unui anumit domeniu, platformă sau metodă de dezvoltare de software.

PROFILE UML

Exemple



Stereotip – clasă din metamodelul UML Profile care definește modul în care este extinsă o metaclassă existentă în UML metamodel.

PROFILE UML

Profile diagram – definește un profil prin adaptările aplicate metamodelului UML de bază; pachet care extinde metamodelul de referință UML.

Stereotype = extinde o metaclasă pentru a-i adăuga noi semantici

Tagged value = meta-atribut standard

Profil = tip specific de pachet

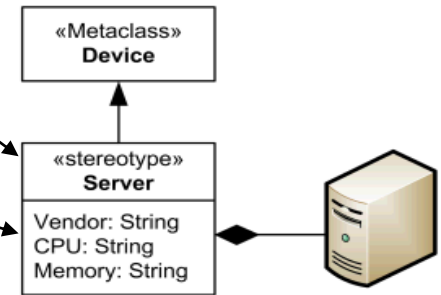


Fig. Extras dintr-o diagramă de profil

Profil :

- Aplicat / retractat dinamic la/de la un model
- Combinat dinamic cu alte profile aplicate simultan aceluiași model
- Poate extinde un alt profil

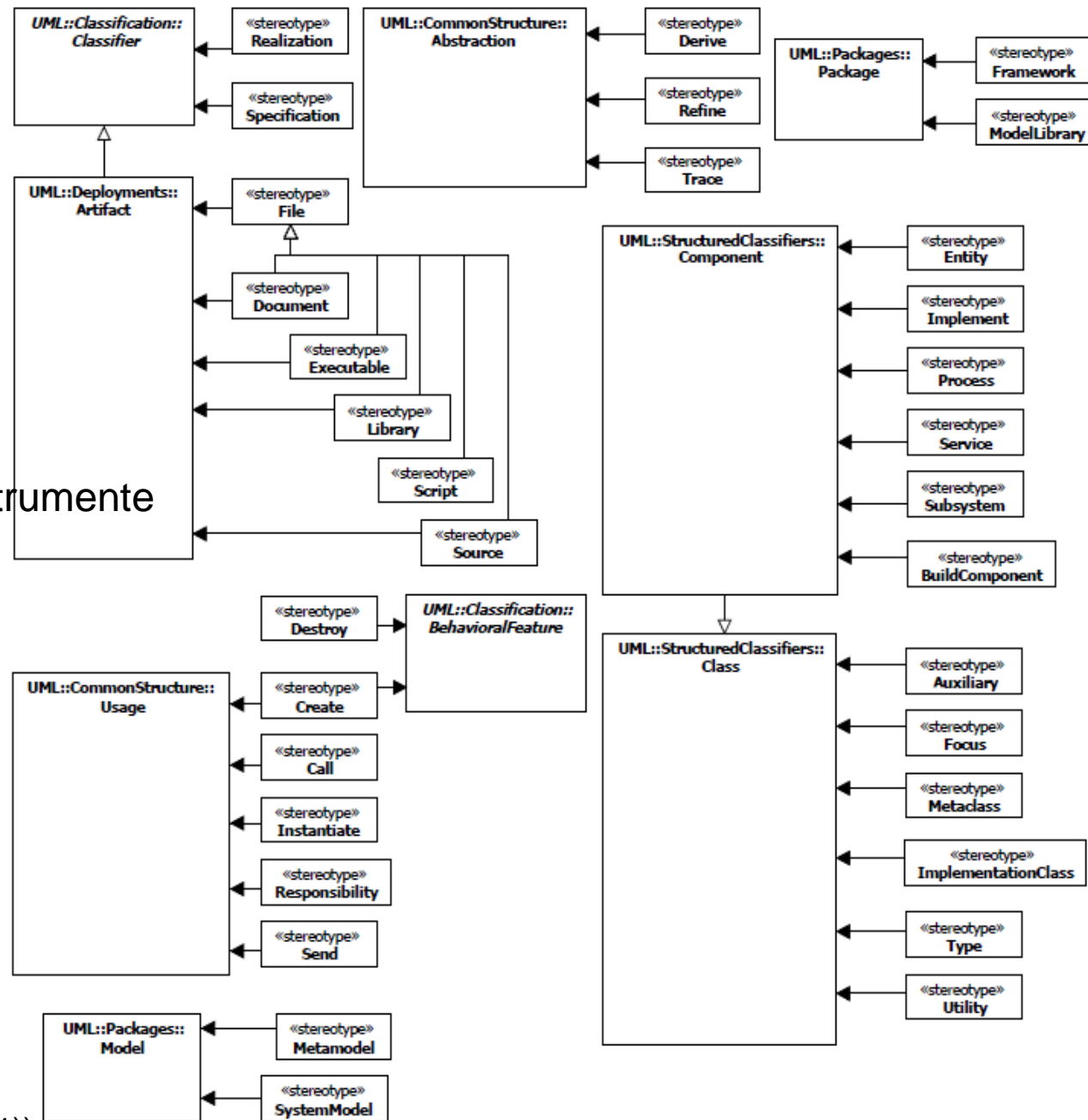
PROFILUL UML

Stereotipurile standard

Specifică un set standard de stereotipuri predefinite.

Cerințe implicate pentru instrumente software de modelare UML:

- implementează toate specificațiile UML
- suportă setul standard de stereotipuri



(specificații UML 2.5.1 pag. 679 (721))

PROFILE UML

Exemple :

- profil CORBA
- profil pentru date distribuite
- profil pentru servicii avansate și integrate de telecomunicații
- profil pentru procese BPMN
- profil pentru MARTE (Modeling and Analysis of Real-time and Embedded Systems)
- profil pentru testare - <https://www.omg.org/spec/UTP2/2.1/PDF/changebar>

www.omg.org/specs

<https://www.omg.org/spec/category/modeling/>

<https://www.omg.org/spec/category/uml-profile/>

Alt exemplu :

[Fisierul Ex ProfilUML SOA](#)

BIBLIOGRAFIE

Tom Pender, **UML Bible**, Ed. John Wiley, 2003

<https://www.omg.org/spec/UML/2.5.1/PDF>

<http://www.uml-diagrams.org>

Cateva exemple :

<http://creately.com/blog/diagrams/uml-diagram-types-examples/>

DISCUȚIE

IoC și Dependency Injection

<https://martinfowler.com/articles/injection.html>

Dependency Inversion Principle, în relație cu IoC și Dependency Injection

<https://martinfowler.com/articles/injection.html>

???Dependency Inversion Principle, în relație cu IoC și Dependency Injection

<https://medium.com/ssense-tech/dependency-injection-vs-dependency-inversion-vs-inversion-of-control-lets-set-the-record-straight-5dc818dc32d1>