Ingineria Programării

Cursul 4 – 13 Martie 2017

adiftene@info.uaic.ro

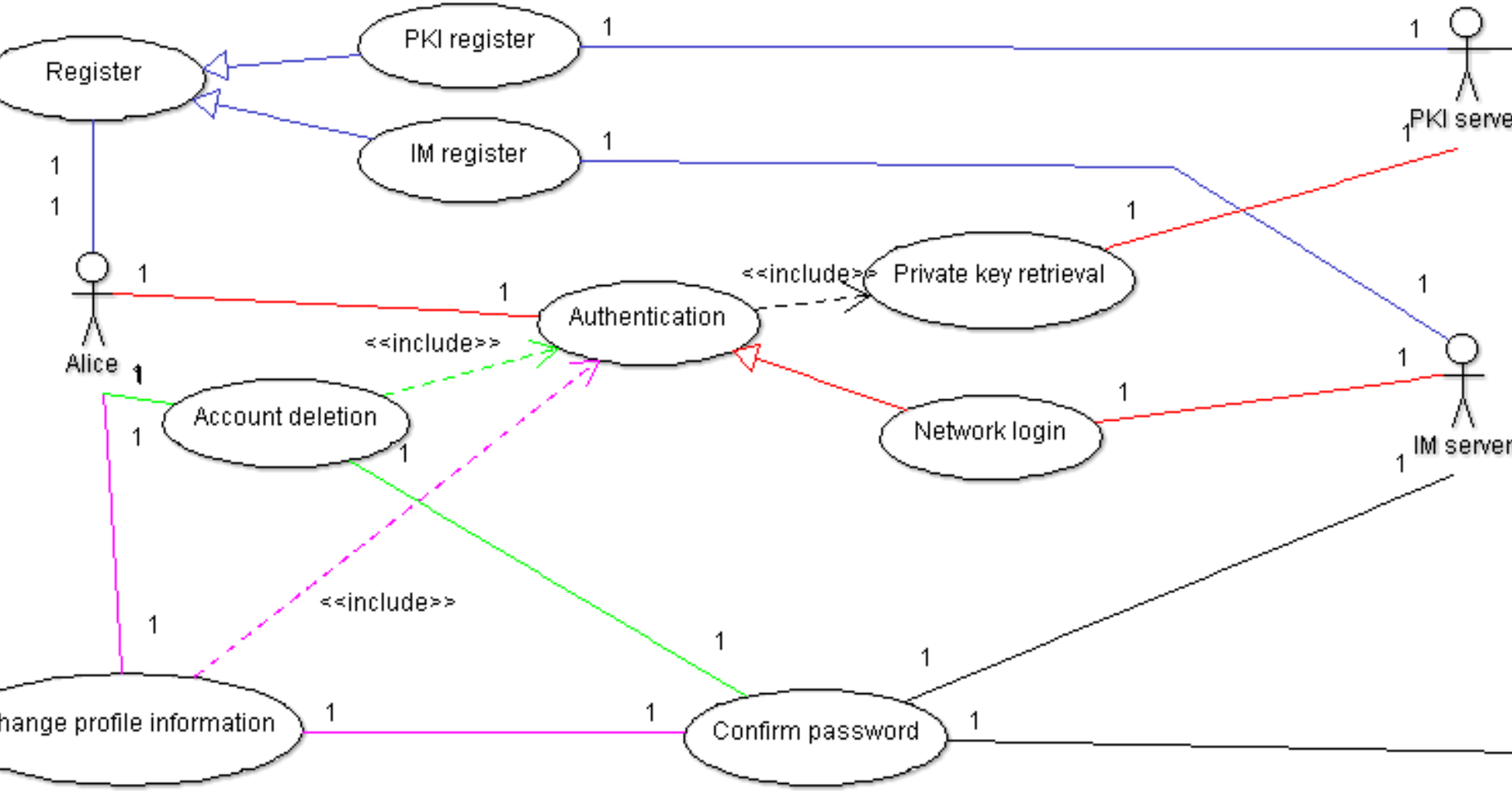
Cuprins

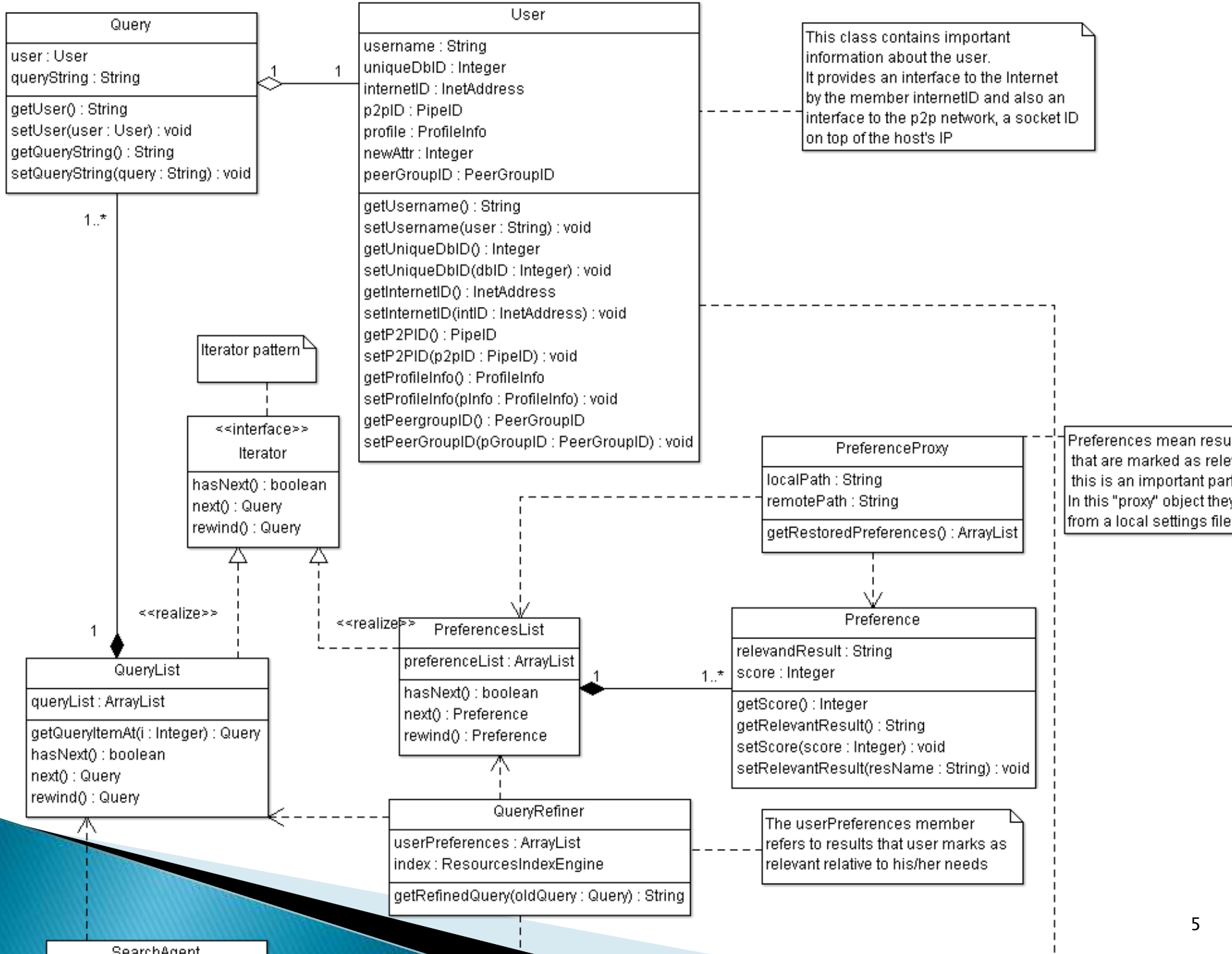
- Din Cursurile trecute…

- Diagrame UML:
  - Interacțiuni (secvență, colaborare)
  - Comportamentale (Stări, Activități)
  - Structură (Deployment)

Din cursurile trecute…

- Diagrame
- Diagrame UML
- Diagrame Use Case
- Diagrame de Clase


Register
PKI register
IM register
PKI serve
Alice
Authentication
<<include>>
Private key retrieval
Account deletion
Network login
IM server
<<include>>
hange profile information
Confirm password
1


Query
user : User
queryString : String
getUser() : String
setUser(user : User) : void
getQueryString() : String
setQueryString(query : String) : void
1
1
User
username : String
uniqueDbID : Integer
internetID : InetAddress
p2pID : PipeID
profile : ProfileInfo
newAttr : Integer
peerGroupID : PeerGroupID
getUsername() : String
setUsername(user : String) : void
getUniqueDbID() : Integer
setUniqueDbID(dbID : Integer) : void
getInternetID() : InetAddress
setInternetID(intID : InetAddress) : void
getP2PID() : PipeID
setP2PID(p2pID : PipeID) : void
getProfileInfo() : ProfileInfo
setProfileInfo(pInfo : ProfileInfo) : void
getPeergroupID() : PeerGroupID
setPeerGroupID(pGroupID : PeerGroupID) : void
This class contains important information about the user.
It provides an interface to the Internet by the member internetID and also an interface to the p2p network, a socket ID on top of the host's IP
1..*
Iterator pattern
<<interface>>
Iterator
hasNext() : boolean
next() : Query
rewind() : Query
PreferenceProxy
localPath : String
remotePath : String
getRestoredPreferences() : ArrayList
Preferences mean resu
that are marked as rele
this is an important part
In this "proxy" object they
from a local settings file
<<realize>>
<<realize>>
1
PreferencesList
preferenceList : ArrayList
hasNext() : boolean
next() : Preference
rewind() : Preference
1
1..*
Preference
relevandResult : String
score : Integer
getScore() : Integer
getRelevantResult() : String
setScore(score : Integer) : void
setRelevantResult(resName : String) : void
QueryList
queryList : ArrayList
getQueryItemAt(i : Integer) : Query
hasNext() : boolean
next() : Query
rewind() : Query
QueryRefiner
userPreferences : ArrayList
index : ResourcesIndexEngine
getRefinedQuery(oldQuery : Query) : String
The userPreferences member refers to results that user marks as relevant relative to his/her needs
SearchAgent

UML2.0 – 13 Tipuri de Diagrame

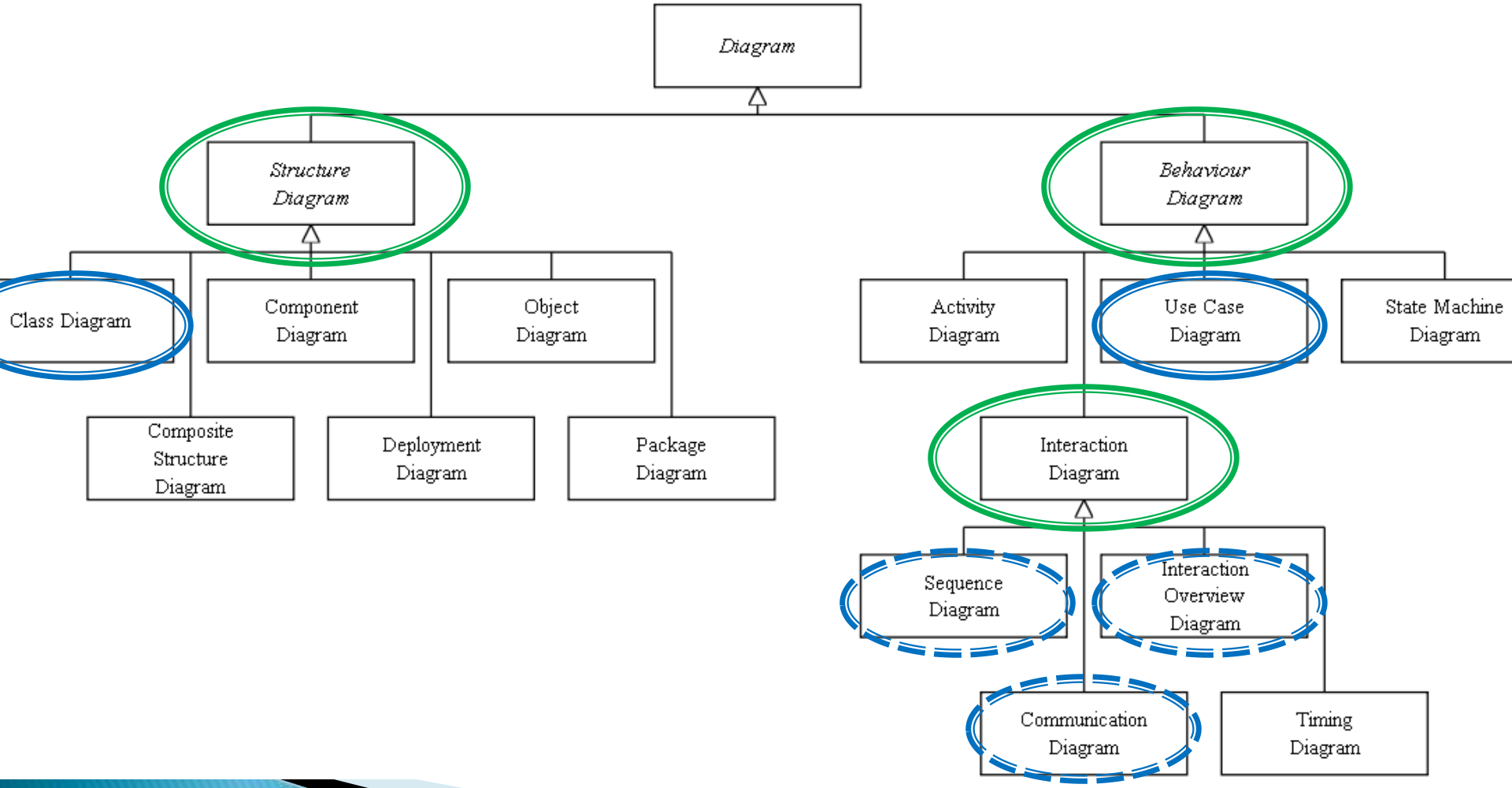
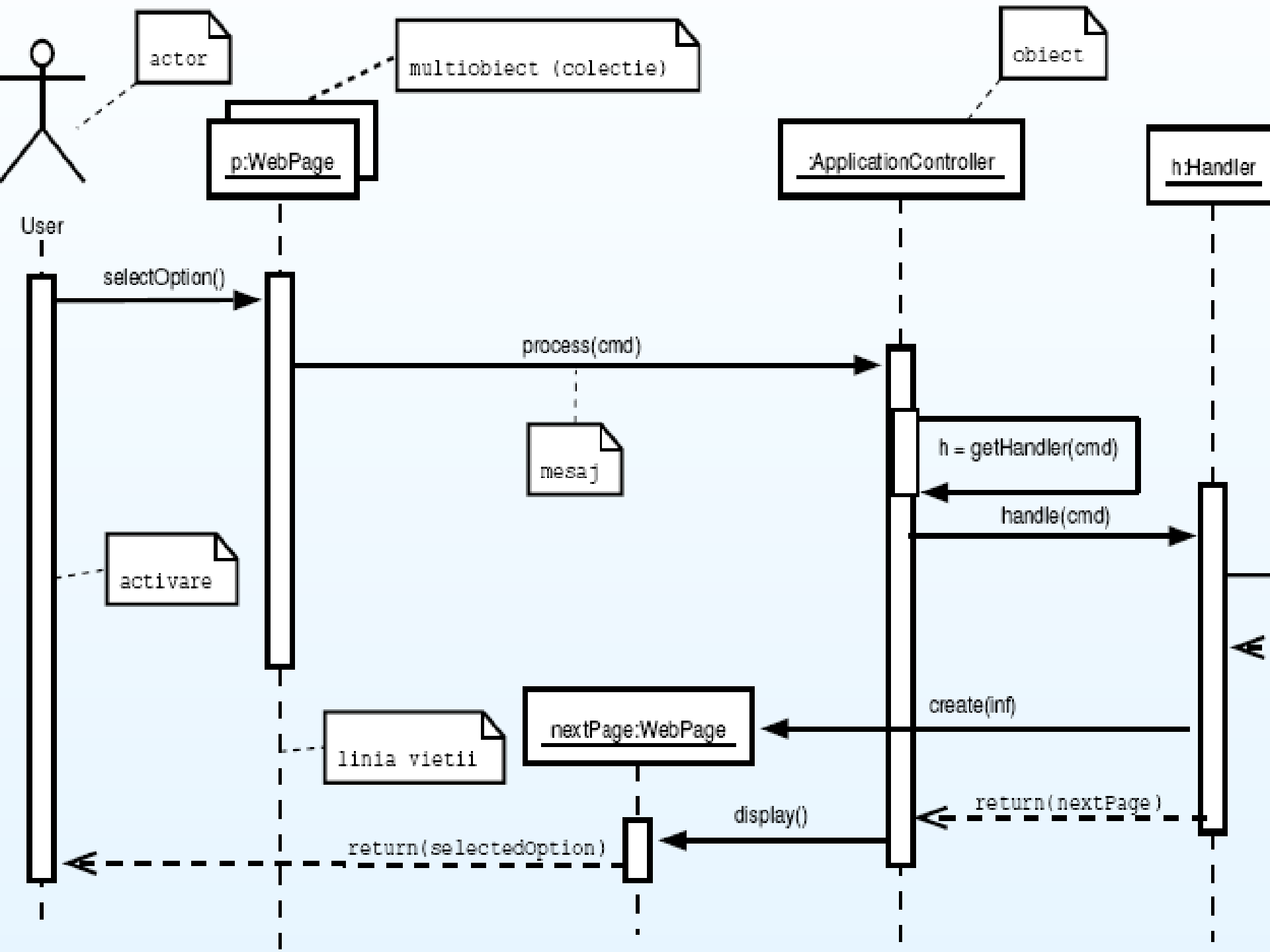
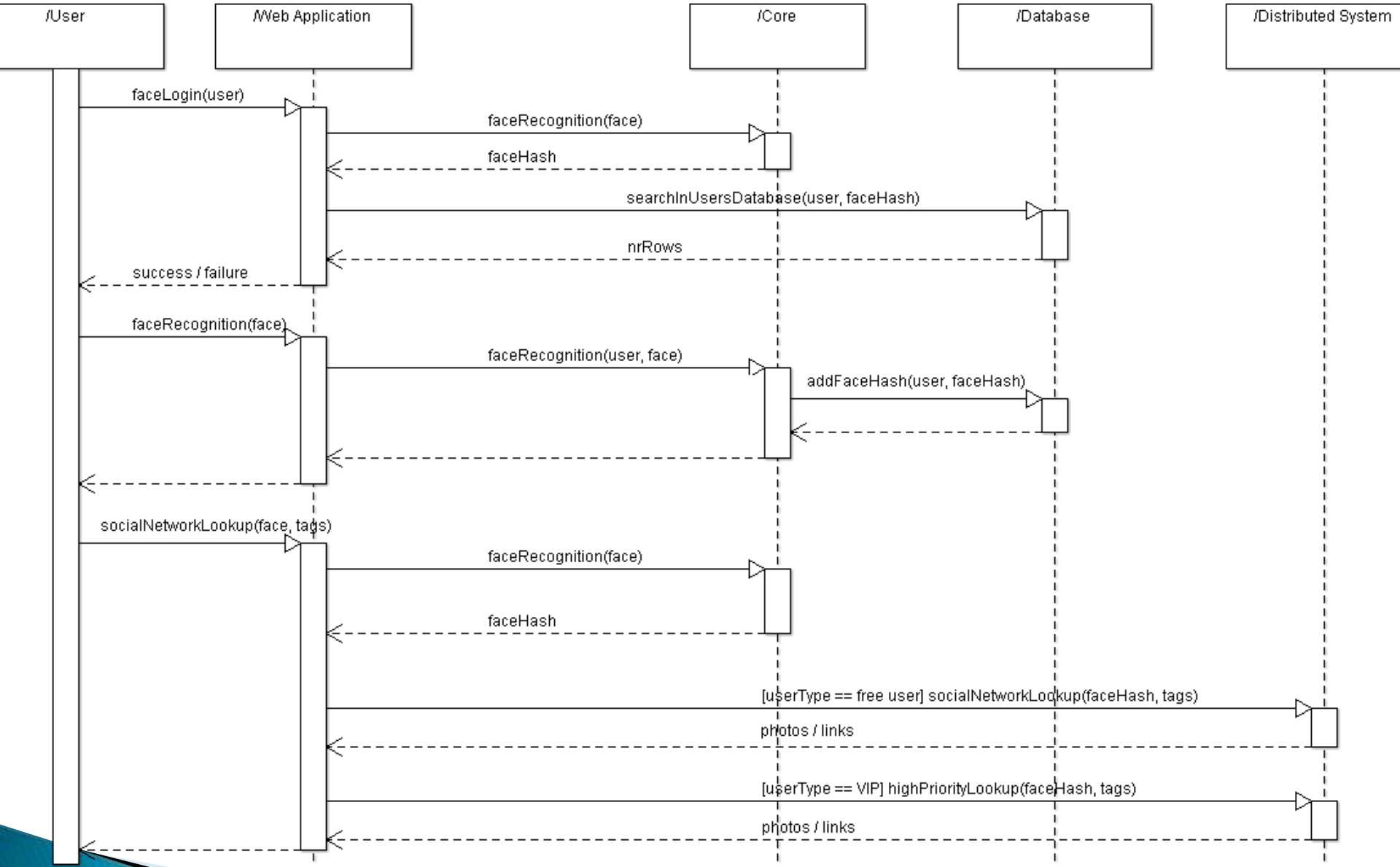


Diagrama de Secvenţă

- **Diagrama de secvenţă** curprinde secvenţa acţiunilor care au loc în sistem, invocarea metodelor fiecărui obiect ca şi ordinea în timp în care aceste invocări au loc

- O diagramă de secvenţă este bidimensională
  - Pe axa verticală se prezintă viaţa obiectului
    - linia vieţii obiectelor (grafic: linie punctată)
    - perioada de activare în care un obiect preia controlul execuţiei (grafic: dreptunghi pe linia vieţii)
  - Pe axa orizontală se arată secvenţa creării sau invocărilor
    - mesaje ordonate în timp (grafic: săgeţi)

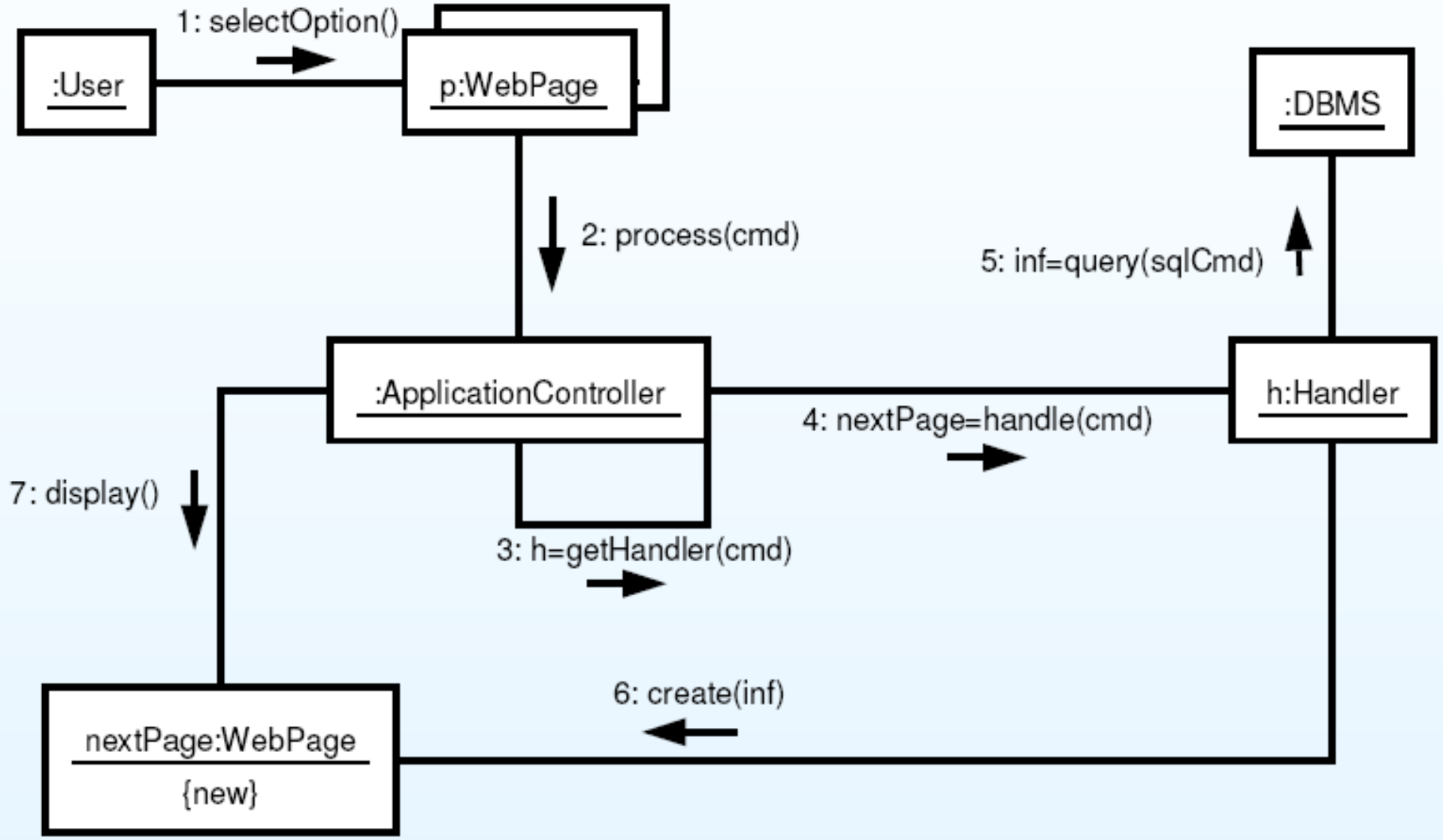

actor
multiobiect (colectie)
obiect
p:WebPage
:ApplicationController
h:Handler
User
selectOption()
process(cmd)
mesaj
h = getHandler(cmd)
handle(cmd)
activare
create(inf)
nextPage:WebPage
linia vietii
return(nextPage)
display()
return(selectedOption)


/User
/Web Application
/Core
/Database
/Distributed System
faceLogin(user)
faceRecognition(face)
faceHash
searchInUsersDatabase(user, faceHash)
nrRows
success / failure
faceRecognition(face)
faceRecognition(user, face)
addFaceHash(user, faceHash)
socialNetworkLookup(face, tags)
faceRecognition(face)
faceHash
[userType == free user] socialNetworkLookup(faceHash, tags)
photos / links
[userType == VIP] highPriorityLookup(faceHash, tags)
photos / links

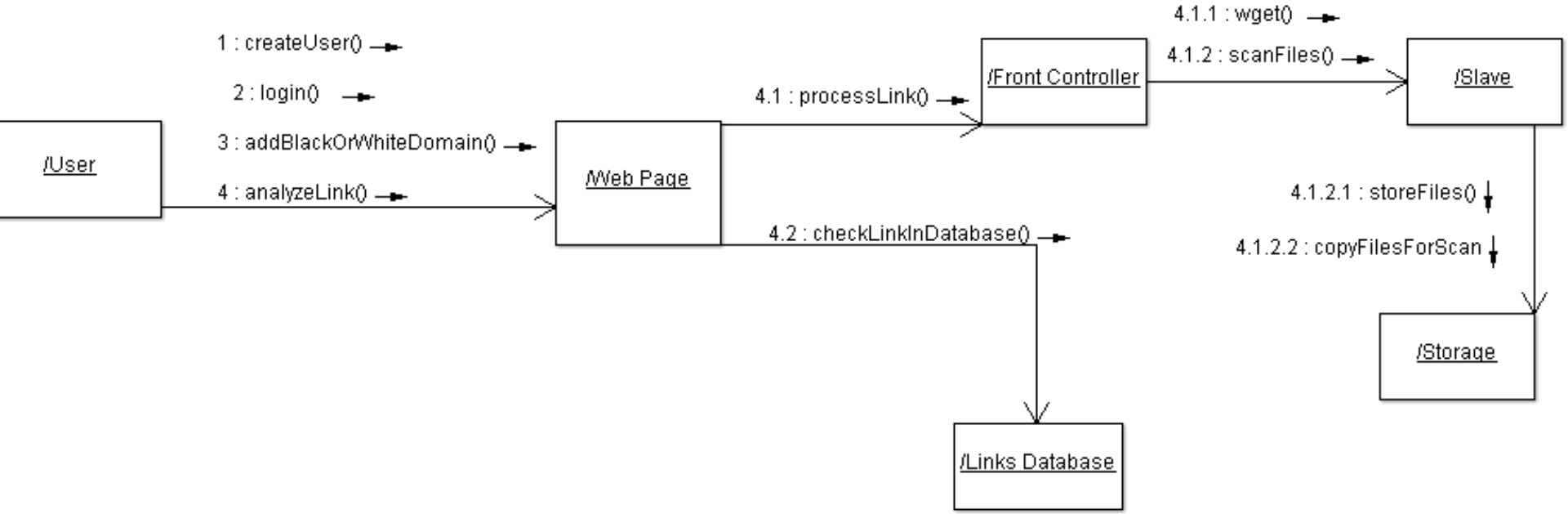
Diagramă de Colaborare

- Pune accentul pe organizarea structurală a obiectelor care participă la interacţiune
- Ilustrează mai bine ramificări complexe, iteraţii şi comportament concurent
- Poate conţine:
  - Obiecte, clase, actori
  - Legături între acestea
  - Mesaje

Exemplul 1


1: selectOption()
:User
p:WebPage
:DBMS
2: process(cmd)
5: inf=query(sqlCmd)
:ApplicationController
h:Handler
4: nextPage=handle(cmd)
7: display()
3: h=getHandler(cmd)
6: create(inf)
nextPage:WebPage
{new}

Exemplul 3

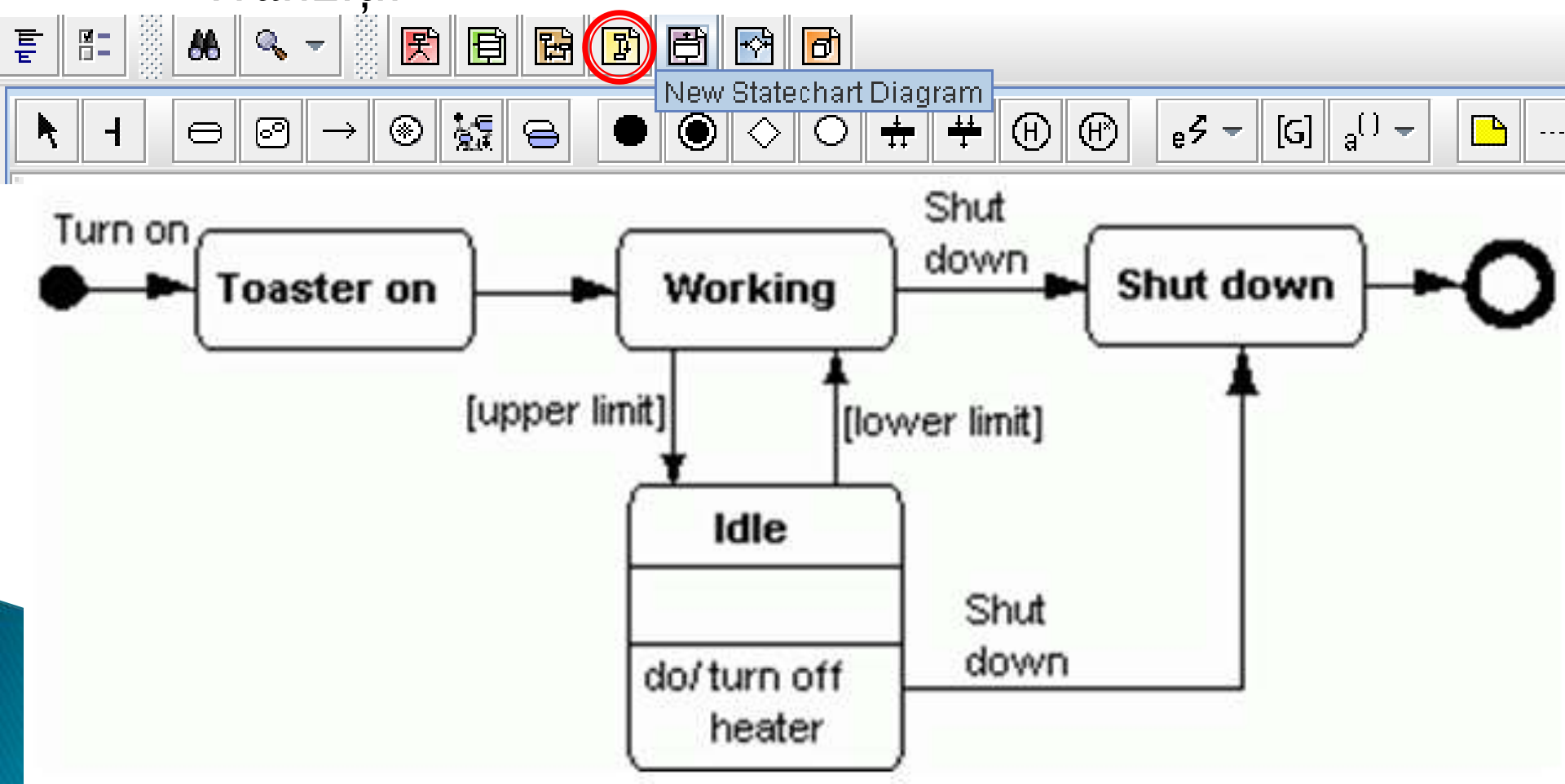

1 : createUser()
2 : login()
3 : addBlackOrWhiteDomain()
4 : analyzeLink()
/User
/Web Page
4.1 : processLink()
/Front Controller
4.1.1 : wget()
4.1.2 : scanFiles()
/Slave
4.1.2.1 : storeFiles()
4.1.2.2 : copyFilesForScan
/Storage
4.2 : checkLinkInDatabase()
/Links Database

Diagrame comportamentale

- Diagrame de stări, diagrame de activități
- Elemente de bază
  - Eveniment
  - Acțiune
  - Activitate

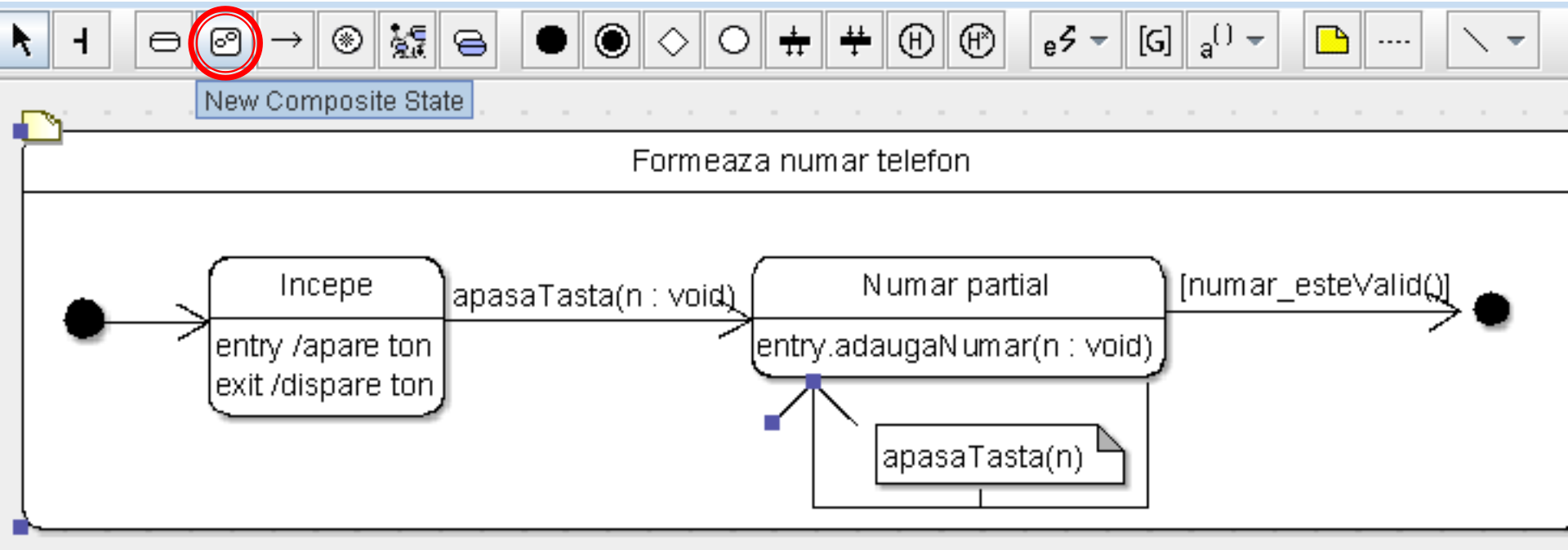
Diagramă de Stări

- Conține:
  - Stări
  - Tranziții



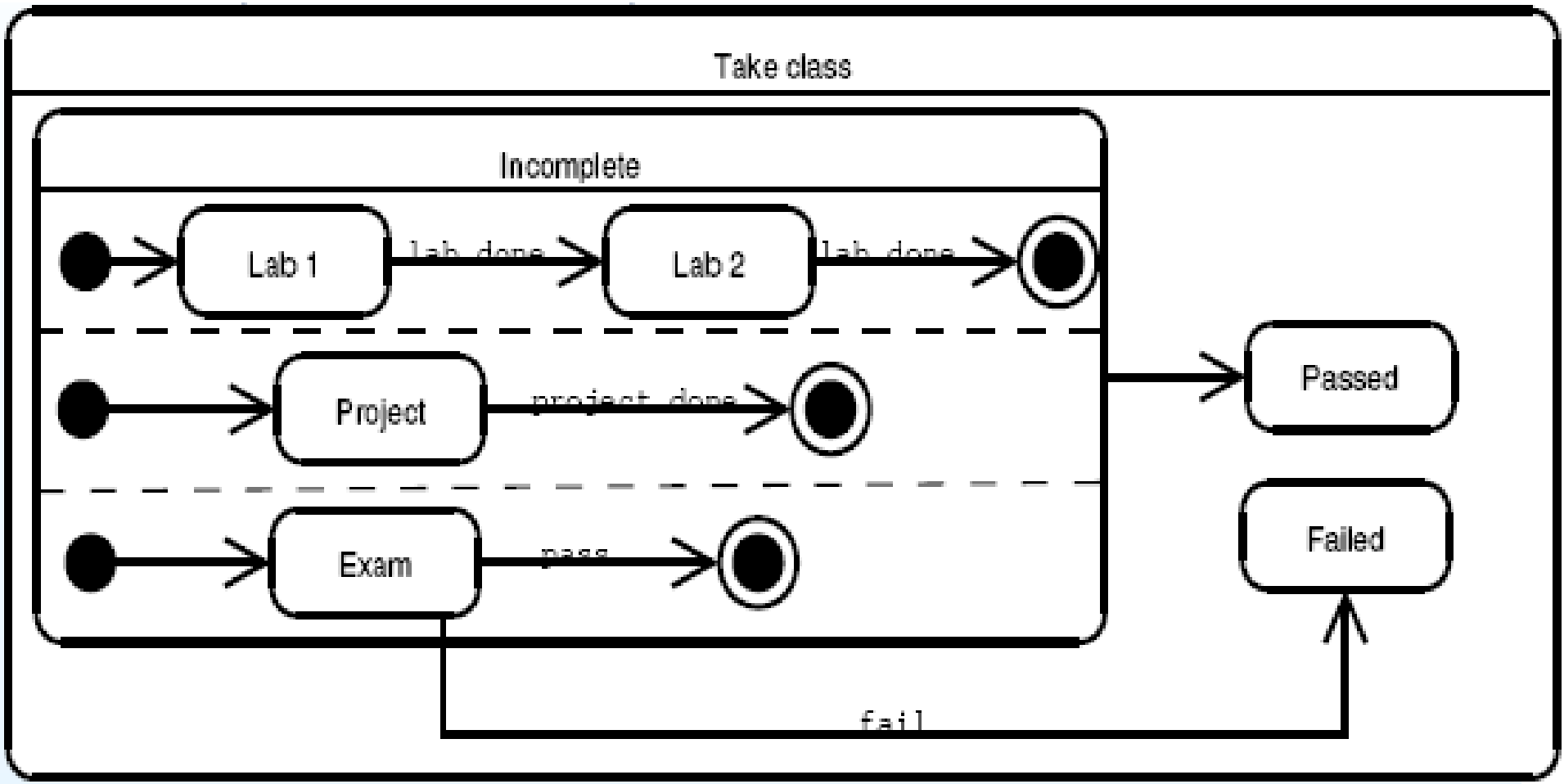
Exemplu de Stare compusă 1

- Stare compusă cu substări **secvențial** active:



Exemplu de Stare compusă 2

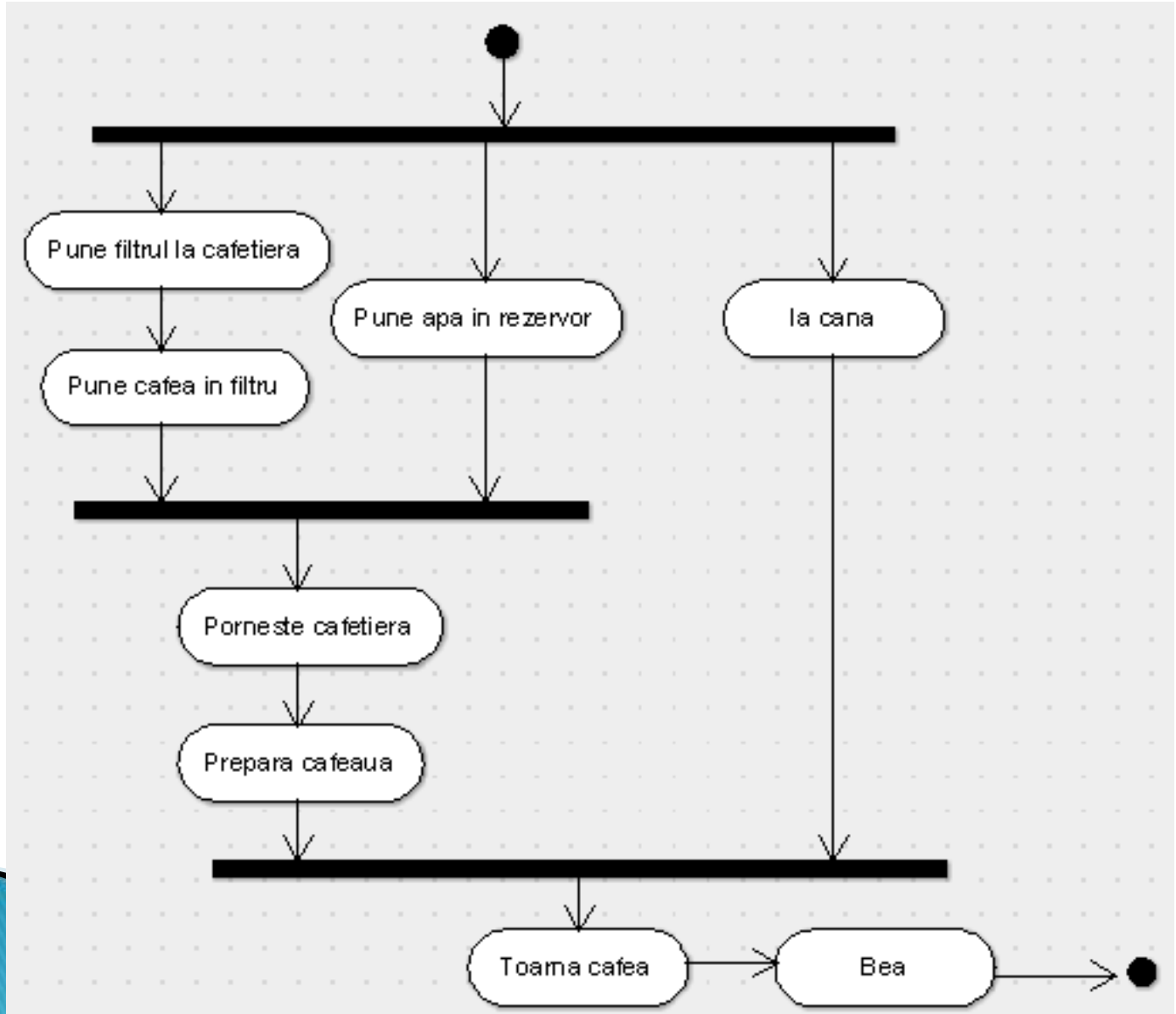
- Stare compusă cu substări **paralel** active:

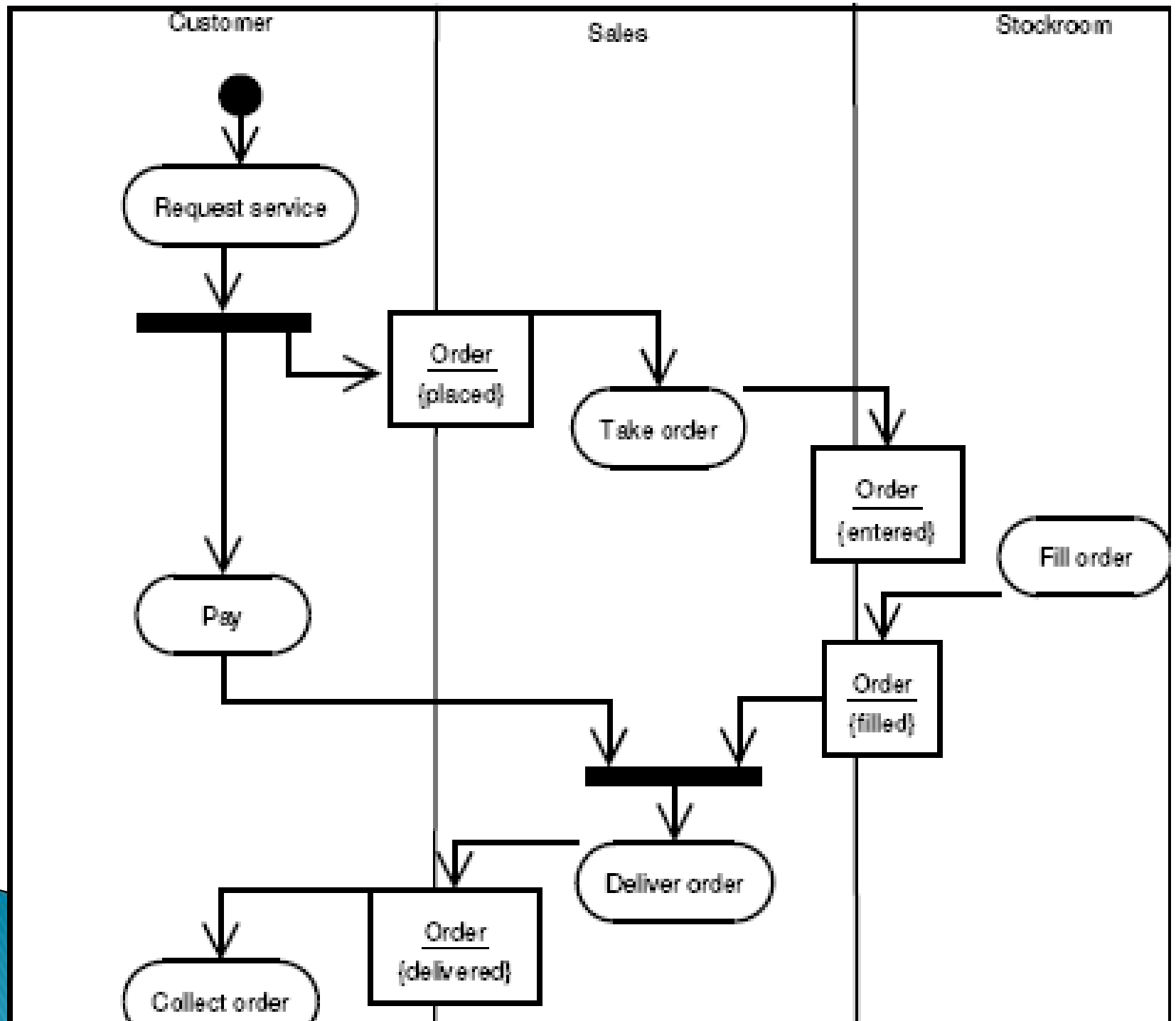


Diagramă de Activități (Activity Diagram)

- Folosită pentru **a modela dinamica unui proces sau a unei operații**
- Evidențiază controlul execuției de la o activitate la alta
- Se atașează:
  - Unei clase (modelează un caz de utilizare)
  - Unui pachet
  - Implementării unei operații

Exemplu de DA (Sincronizare)

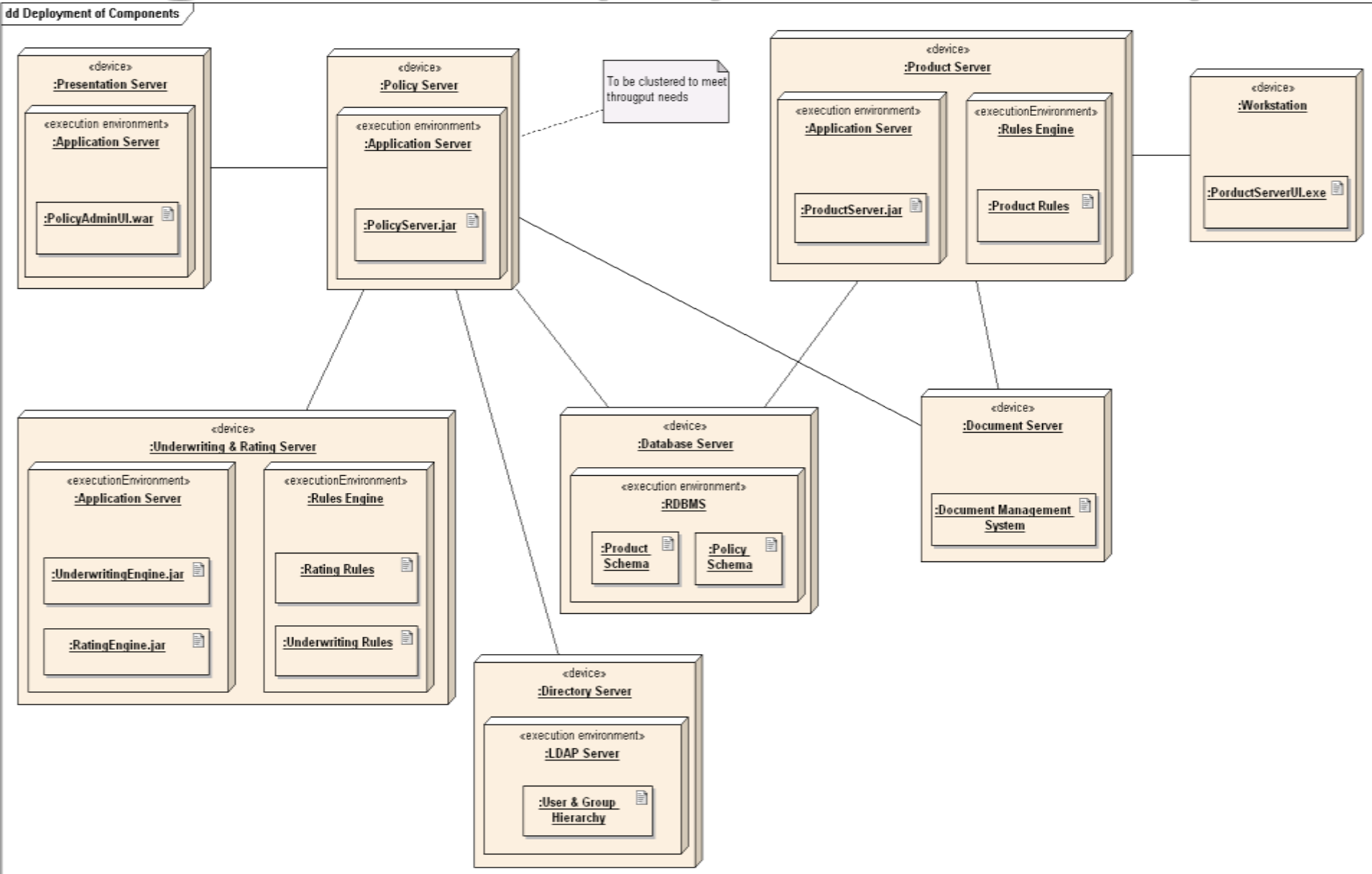



Customer
Sales
Stockroom
Request service
Order
{placed}
Take order
Order
{entered}
Fill order
Pay
Order
{filled}
Deliver order
Order
{delivered}
Collect order

Diagrame de deployment

- Modelează mediul hardware în care va funcționa proiectul
- Exemplu: pentru a descrie un site web o diagramă de deployment va conține **componentele hardware**
  - server-ul web,
  - server-ul de aplicații,
  - server-ul de baze de date
- **Componentele software** de pe fiecare din acestea
  - Aplicația web
  - Baza de date
- Modul în care acestea sunt conectate:
  - JDBC, REST, RMI

Diagramă de deployment – Exemplu 1



Diagrame de Pachete (Package Diagram)

- Pachetul:
  - Este un container logic pentru elemente între care se stabilesc legături
  - Definește un spațiu de nume
  - Toate elementele UML pot fi grupate în pachete (cel mai des pachetele sunt folosite pentru a grupa clase)
  - Un pachet poate conține subpachete => se creează o structură arborescentă (similară cu organizarea fișierele/directoarelor)

Diagrame de Pachete 2

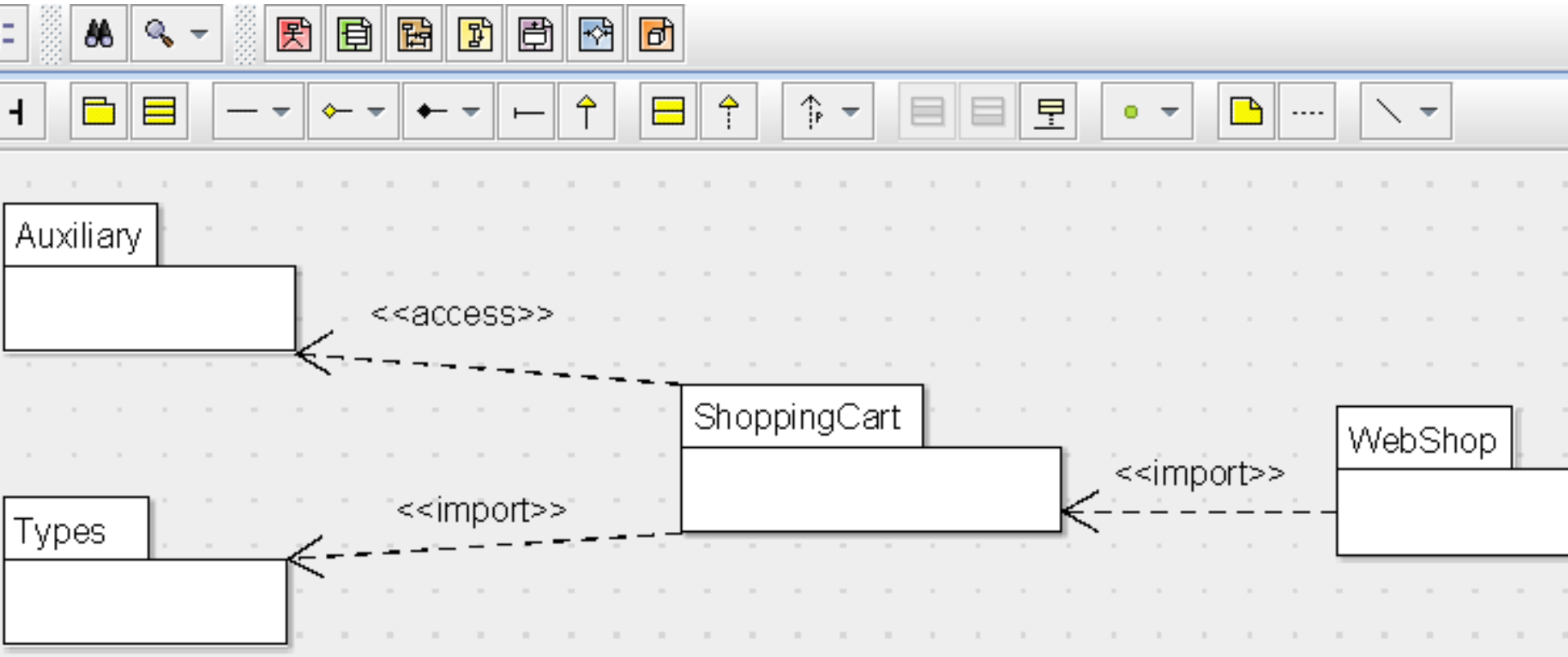
- Relații:
  - dependență <<access>> = import privat
  - dependență <<import>> = import public

- Ambele relații permit folosirea elementelor aflate în pachetul destinație de către elementele aflate în pachetul sursă fără a fi necesară calificarea numelor elementelor din pachetul destinație (similar directivei **import** din java)

- Aceste tipuri de diagrame se realizează în cadrul diagramelor de clasă

Exemplu de Diagramă de Pachete

- Elementele din Types sunt importate în ShoppingCart și apoi sunt importate mai departe de către WebShop
- Elementele din Auxiliary pot fi accesate însă doar din ShoppingCart și nu pot fi referite folosind nume necalificate din WebShop



Utilitatea diagramelor de pachete

- Împart sisteme mari în subsisteme mai mici și mai ușor de gestionat
- Permit dezvoltare paralelă iterativă
- Definirea unor interfețe clare între pachete promovează refolosirea codului (ex. pachet care oferă funcții grafice, pachet care oferă posibilitatea conectării la BD, etc...)

Recomandări în realizarea diagramelor UML

- Diagramele să nu fie nici prea complicate, dar nici prea simple: scopul este comunicarea eficientă
- Dați nume sugestive elementelor componente
- Aranjați elementele astfel încât liniile să nu se intersecteze
- Încercați să nu arătați prea multe tipuri de relații odată (evitați diagramele foarte complicate)
- Dacă este nevoie, realizați mai multe diagrame de același tip

Concluzii

- Diagrame UML:
  - Interacțiuni
  - Comportamentale
  - Structură

Bibliografie

- Ovidiu Gheorghieş, Curs 5 IP
- www.uml.org