

## *Curs 2*

### *Specificarea modelelor folosind UML*

*Suport de curs bazat pe B. Bruegge and A.H. Dutoit*

*"Object-Oriented Software Engineering using UML, Patterns, and Java"*

## Sumar Curs 2

---

- Notății/limbaje. UML
- Tipuri de modele ale sistemelor soft
- Diagrame de cazuri de utilizare
- Diagrame de clase/obiecte
- Diagrame de interacțiune
- Diagrame de tranziție a stărilor
- Diagrame de activități

# Notatii / limbaje. UML

---

- *O notație (un limbaj)* reprezintă o mulțime de reguli, textuale sau grafice, folosite pentru specificarea modelelor
- În cadrul unui proiect soft, o notație este un instrument de comunicare (de idei, decizii de analiză, proiectare, etc.)
- Pentru a permite o comunicare eficientă, o notație trebuie
  - să aibă o semantică bine definită
  - să fie adecvată reprezentării acelor aspecte pentru care este folosită
  - să fie bine înțeleasă de către toți participanții la proiect
- *UML (Unified Modeling Language)*
  - limbajul standard adoptat de industrie pentru reprezentarea modelelor orientate obiect
  - a rezultat prin unificarea notațiilor utilizate de metodologiile
    - OMT (Object Modeling Technique - [Rumbaugh et al., 1991])
    - Booch ([Booch, 1994])
    - OOSE (Object Oriented Software Engineering - [Jacobson et al., 1992])
  - oferă un spectru larg de notații pentru reprezentarea diferitor tipuri de sisteme și a diferitor aspecte/vederi ale unui sistem

# Tipuri de modele ale sistemelor soft

---

- Modelul funcțional (eng. *functional model*)
  - descrie funcționalitatea sistemului din perspectiva utilizatorului
  - reprezentat în UML folosind *diagrame de cazuri de utilizare*
- Modelul obiectual (structural) (eng. *object model*)
  - descrie structura sistemului în termeni de clase, attribute, asocieri și operații
  - reprezentat în UML folosind *diagrame de clase*
  - evoluția modelului obiectual
    - *modelul obiectual de analiză sau modelul conceptual* (eng. *analysis object model*) - descrie conceptele din domeniul problemei relevante pentru sistemul studiat
    - *modelul obiectual corespunzător proiectării de sistem* (eng. *system design object model*) - rafinare a modelului conceptual, ce include descrieri ale interfețelor subsistemelor
    - *modelul obiectual de proiectare* (eng. *object design model*) - rafinare a modelului obiectual aferent proiectării de sistem, incluzând descrierea detaliată a obiectelor din domeniul soluției
- Modelul dinamic (eng. *dynamic model*)
  - descrie comportamentul intern al sistemului
  - reprezentat în UML folosind
    - *diagrame de interacțiune* - secvențe de mesaje schimbate între obiecte
    - *diagrame de tranziție a stărilor* - stările obiectelor și tranzițiile între stări
    - *diagrame de activități* - fluxuri de date și de control

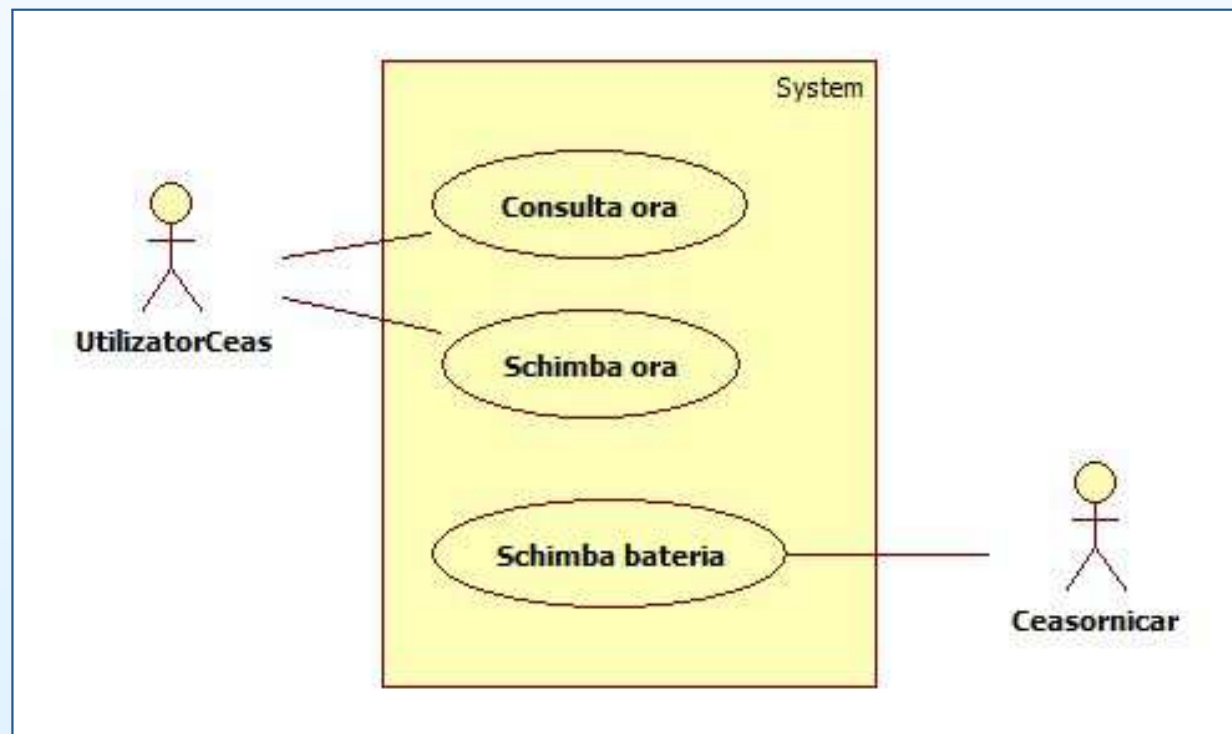
# Diagrame de cazuri de utilizare

---

- *Cazurile de utilizare* (eng. *use cases*) sunt folosite în cadrul activităților de colectare și analiză a cerințelor, pentru a reprezenta funcționalitățile sistemului
  - Cazurile de utilizare surprind comportamentul sistemului din perspectiva utilizatorilor externi
  - Un caz de utilizare descrie o funcție oferită de către sistem, care are rezultate tangibile pentru un actor
- *Actorii* (eng. *actors*) reprezintă roluri jucate de entități externe sistemului, care interacționează cu acesta
  - utilizatori umani (administrator, client ATM, abonat bibliotecă)
  - alte sisteme soft, echipamente hardware, etc.
- Identificarea actorilor și a cazurilor de utilizare permite definirea *frontierei* sistemului (eng. *system boundary / subject*), mai precis diferențierea între sarcinile îndeplinite de sistem și cele îndeplinite de mediul său
  - Actorii sunt înafara frontierei, cazurile de utilizare sunt înăuntrul acesteia

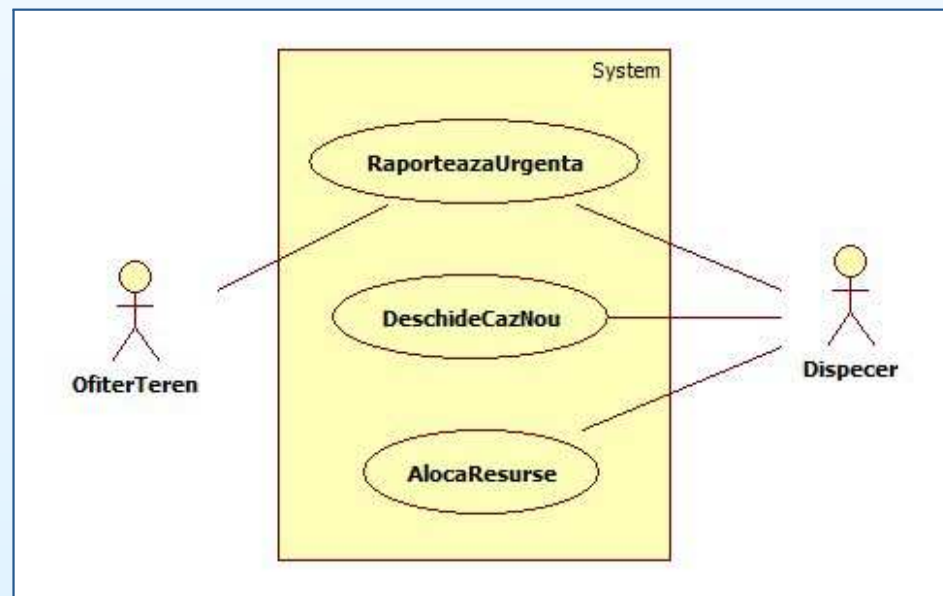
## Diagrame de cazuri de utilizare (cont.)

- Ex.: diagramă a cazurilor de utilizare descriind funcționalitatea unui ceas simplu
  - Sintaxa UML: actor - eng. *stick figure*, caz de utilizare - elipsă, frontiera sistemului - dreptunghi (eng. *box*), etichetate cu denumirile aferente
  - ! Cazurile de utilizare se denumesc cu expresii verbale sugestive pentru funcționalitatea oferită



## Diagrame de cazuri de utilizare (cont.)

- Ex.: diagramă de cazuri de utilizare aferentă unui sistem de gestiune a accidentelor (SGA)
  - Un ofițer din teren (de poliție/pompieri) are posibilitatea invocării cazului de utilizare *RaporteazaUrgenta*, pentru a notifica un dispecer relativ la un nou incident. Ca și răspuns, dispecerul invocă *DeschideCazNou*, pentru a iniția gestiunea incidentului. Dispecerul introduce în baza de date informațiile preliminare primite de la ofițer și alocă resurse (mașini, resursa umană), prin intermediul cazului de utilizare omonim.



# Cazuri de utilizare

---

- Conținutul unui caz de utilizare poate fi descris textual, folosind un șablon cu următoarele elemente
  - **Nume** - numele cazului de utilizare, unic în sistem
  - **Actori participanți** - actorii care comunică cu acel caz de utilizare
  - **Flux de evenimente** - secvența de interacțiuni între actori și sistem, care definește cazul de utilizare. Fluxul normal și fluxurile alternative (erori, condiții speciale) se vor descrie separat, pentru claritate
  - **Condiții de intrare (precondiții)** - condiții care trebuie satisfăcute anterior inițierii cazului de utilizare
  - **Condiții de ieșire (postcondiții)** - condiții ce trebuie satisfăcute după finalizarea cazului de utilizare
  - **Cerințe de calitate** - constrângeri privind performanța sistemului, implementarea lui, platforma hardware folosită, etc.
- În cadrul șablonului, descrierea cazului de utilizare se face în limbaj natural, susținând comunicarea facilă cu clienții și utilizatorii sistemului



## Cazuri de utilizare (cont.)

- Ex.: Descriere textuală a cazului de utilizare *RaporteazăUrgență*

<b>Nume</b>	<i>RaporteazăUrgență</i>
<b>Actori</b>	Inițiat de <i>OfițerTeren</i> Comunică cu <i>Dispecerul</i>
<b>Flux de evenimente</b> (scenariu normal)	<ol style="list-style-type: none"><li>1. <i>Ofițerul</i> activează funcția <i>Raportează urgență</i> a terminalului.</li><li>2. Sistemul SGA afișează un formular <i>Ofițerului</i>.</li><li>3. <i>Ofițerul</i> completează formularul, inserând nivelul de alertă, tipul, locația și o scurtă descriere a situației. Propune și posibile soluții la situația de urgență. După completare, <i>Ofițerul</i> trimite formularul.</li><li>4. Sistemul primește formularul și notifică <i>Dispecerul</i>.</li><li>5. <i>Dispecerul</i> consultă informația primită și apelează cazul de utilizare <i>DeschideCazNou</i>. <i>Dispecerul</i> optează pentru una dintre soluțiile propuse și confirmă primirea formularului.</li><li>6. Sistemul afișează confirmarea și soluția aleasă <i>Ofițerului</i>.</li></ol>
<b>Condiții de intrare</b>	<i>Ofițerul</i> este logat în sistem.
<b>Condiții de ieșire</b>	<i>Ofițerul</i> a primit confirmarea de la <i>Dispecer</i> SAU un mesaj referitor la o eroare de comunicare.
<b>Cerințe de calitate</b>	Confirmarea <i>Dispecerului</i> ajunge în maxim 30 de sec. după trimitere.

# Scenarii

---

- Un caz de utilizare este o abstractizare ce acoperă toate scenariile posibile aferente funcționalității descrise
- Un *scenariu* este o instanță a unui caz de utilizare, ce descrie o secvență concretă de acțiuni/evenimente
  - scenariile sunt exemple ce ilustrează situații tipice - sunt focusate pe inteligibilitate
  - cazurile de utilizare sunt folosite pentru a surprinde toate situațiile posibile - sunt focusate pe completitudine
- Scenariile pot fi descrise folosind un șablon cu trei câmpuri
  - **Nume scenariu** - numele scenariului, pentru o referire neambiguă (subliniat, pentru a indica faptul că e o instanță)
  - **Instanțele actorilor participanți** (sublinate)
  - **Fluxul de evenimente**


## Scenarii (cont.)

- Ex.: Scenariul *IncendiereDepozit* al cazului de utilizare *RaporteazăUrgență*

<b>Nume</b>	<u>IncendiereDepozit</u>
<b>Instanțe</b>	<u>bob, alice : OfițerTeren</u>
<b>actori</b>	<u>john : Dispecer</u>
<b>Flux de evenimente</b>	<ol style="list-style-type: none"><li>1. Trecând prin dreptul unui depozit, Bob simte miros de fum. Partenera sa, Alice, activează funcția <i>Raportează urgență</i> pe terminalul SGA.</li><li>2. Alice introduce adresa clădirii, o scurtă descriere a locației curente și un nivel de alertă. Zona fiind aglomerată, solicită o echipă de pompieri și mai multe de medici. Trimite formularul și așteaptă confirmarea dispecerului.</li><li>3. John, dispecerul, este alertat de un semnal sonor al stației sale de lucru. Citește informațiile trimise de Alice și confirmă primirea lor. Alocă o echipă de pompieri și două de medici și îi trimite lui Alice ora estimată a sosirii acestora.</li><li>5. Alice primește confirmarea și estimarea.</li></ol>

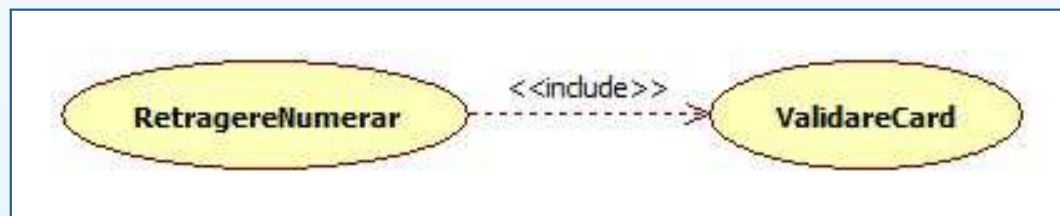
# Relații

---

- Relații posibile în diagramele de cazuri de utilizare
  - comunicare
  - incluziune
  - extindere
  - generalizare
- *Relația de comunicare* (eng. *communication relationship*)
  - un caz de utilizare și un actor comunică atunci când între aceștia există schimb de informație
  - Ex.: Actorii *Dispecer* și *OfițerTeren* comunică cu cazul de utilizare *RaporteazaUrgenta*; doar *Dispecer* comunică cu *DeschideCazNou* și *AlocaResurse*
  - comunicarea dintre un actor și un caz de utilizare se reprezintă ca o asociere UML binară, 
- *Relația de incluziune* (eng. *include relationship*)
  - reprezintă o dependență între două cazuri de utilizare, semnificația fiind inserarea comportamentului descris de cazul de utilizare inclus în cadrul comportamentului descris de cazul de utilizare care include/de bază (localizarea exactă a acestei inserări se face la nivelul descrierii textuale a cazului care include)

## Relații (cont.)

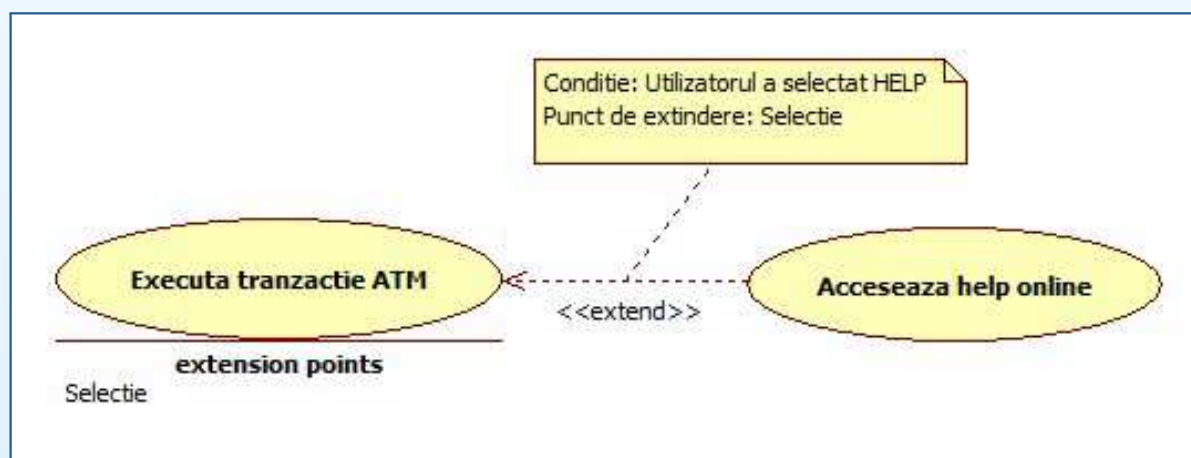
- cazul de utilizare inclus nu este optional, execuția sa este obligatorie pentru finalizarea cu succes a cazului care include (analogie cu apelul de subprogram); ca urmare, sensul dependenței este de la cazul care include către cel inclus
- permite reducerea complexității și eliminarea redundanțelor prin factorizarea secvențelor de interacțiuni comune mai multor cazuri de utilizare
- Ex.: În cadrul unui sistem de tip ATM, cazul de utilizare *RetragereNumerar* include cazul de utilizare *ValidareCard* (definit independent)



- *Relația de extindere (eng. extend relationship)*
  - reprezintă o dependență între două cazuri de utilizare, care precizează *când* (în ce condiții) și *unde* anume poate fi inserat comportamentul aferent cazului de utilizare care extinde în cadrul comportamentului descris de cazul de utilizare extins/de bază

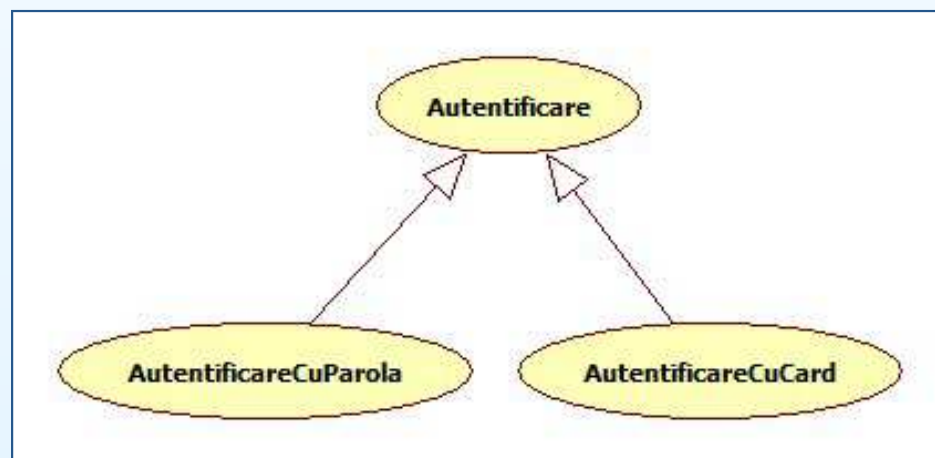
## Relații (cont.)

- cazul de utilizare extins este definit independent și nu depinde de cel care extinde (este de sine stătător)
- cazul de utilizare care extinde poate să nu aibă semnificație de sine stătătoare (poate defini doar un increment comportamental care extinde unul sau mai multe alte cazuri de utilizare, în anumite condiții)
- sensul dependenței este de la cazul care extinde către cel extins
- extinderea are loc la unul sau mai multe puncte de extindere definite în cazul de utilizare de bază (prin nume și variante de localizare)
- Ex.: Cazul de utilizare *Accesează help online* extinde cazul *Execută tranzacție ATM*



## Relații (cont.)

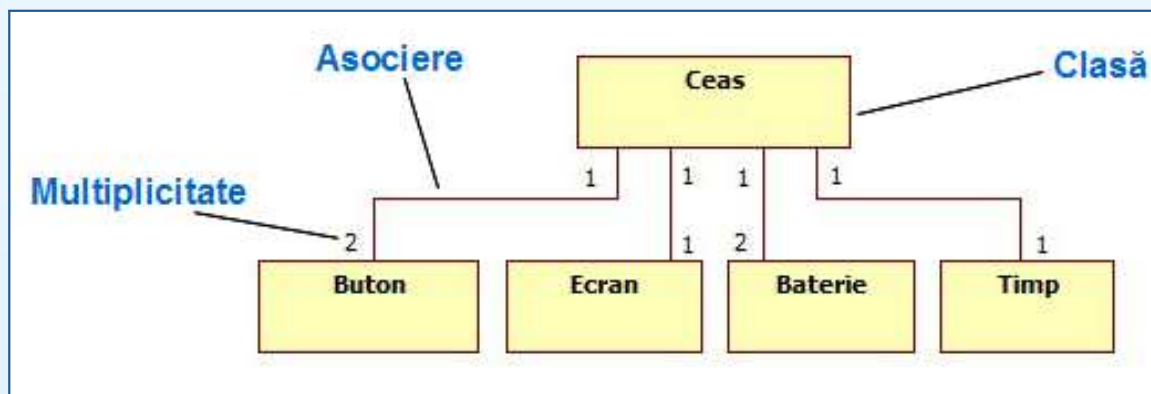
- *Relația de generalizare* (eng. *generalization relationship*)
  - între cazuri de utilizare sau între actori, cu semantica uzuală
  - un caz de utilizare/actor poate specializa un altul, mai general, prin adăugarea de detalii suplimentare
  - Ex.: un caz de utilizare *Autentificare*, identificat inițial în faza de analiză a cerințelor, poate fi ulterior rafinat, rezultând două noi cazuri de utilizare *AutentificareCuParola* și *AutentificareCuCard*, specializări ale primului



- În reprezentarea textuală, cazurile specializate moștenesc actorul ce inițiază interacțiunea, precum și condițiile de intrare și ieșire de la cazul general

# Diagrame de clase/obiecte

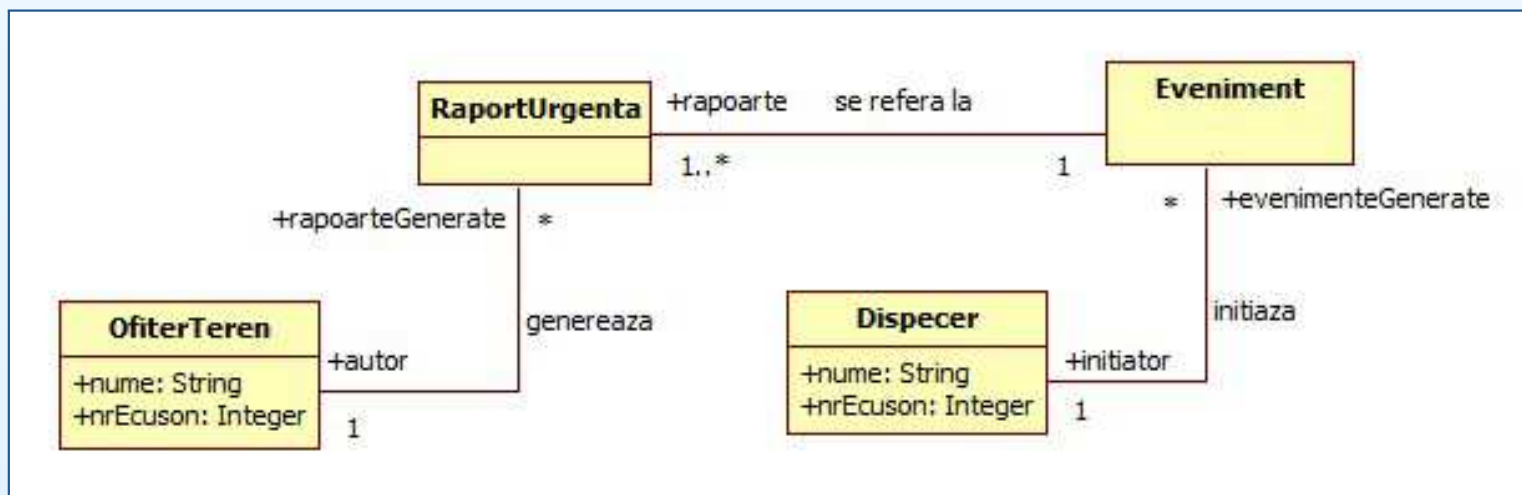
- *Diagramele de clase/obiecte* (eng. *class/object diagrams*) sunt utilizate pentru descrierea structurii unui sistem
  - *Clasele* sunt abstractizări ce specifică structura și comportamentul comune unor mulțimi de obiecte
  - *Obiectele* sunt instanțe ale claselor, create, modificate și distruse pe parcursul execuției unui sistem
    - Un obiect este caracterizat prin *starea* sa, ce include valorile atributelor sale și legăturile cu alte obiecte
    - Fiecare obiect are o *identitate* - poate fi referit individual și diferențiat de alte obiecte
- Ex.: diagramă de clase descriind structura unui ceas simplu





# Clase și obiecte

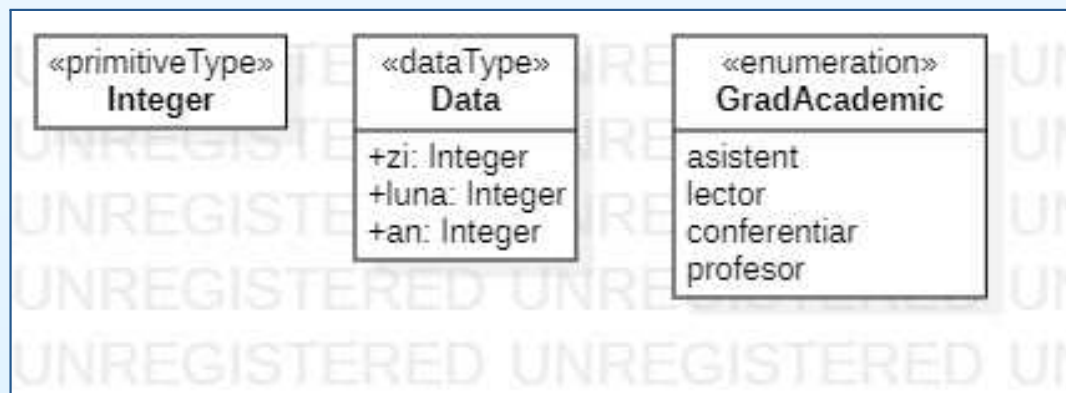
- *Clase* (eng. *classes*)
  - Sunt reprezentate în UML folosind dreptunghiuri cu 3 compartimente (primul compartiment - numele, al doilea - attributele, ultimul - operațiile)
  - Compartimentele pentru attribute și operații pot fi omise
  - Convenție: numele unei clase este un substantiv la singular, prima literă fiind majusculă
  - Attributele sunt caracterizate prin *tip*, operațiile prin *signatură*
- Ex.: diagramă de clase ilustrând entitățile implicate în cazul de utilizare *RaporteazăUrgență* și asocierile între acestea



# Clase și obiecte

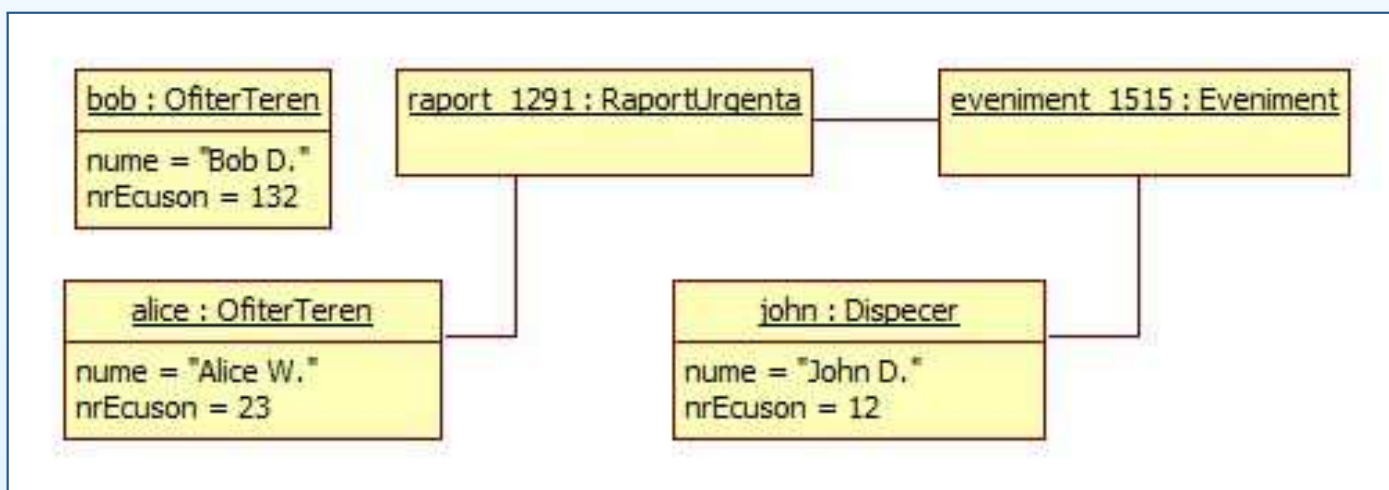
- *Data Type*

- Spre deosebire de instanțele unei clase, instanțele unui *data type* sunt identificate doar prin valoare (nu au identitate proprie, toate instanțele cu aceeași valoare sunt considerate identice)
- Clasificare
  - primitive (nestructurate) - stereotip «primitive»
    - Primitive UML predefinite: Integer, UnlimitedNatural, Real, String, Boolean
  - enumerări (mulțimea valorilor definită ca o listă de literal) - stereotip «enumeration»
  - structurate (au attribute) - stereotip «dataType»



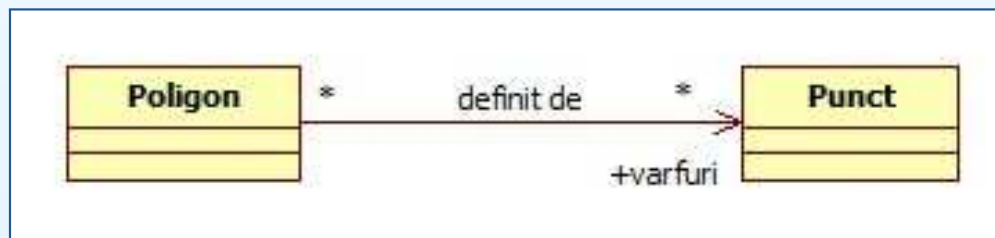
## Clase și obiecte (cont.)

- *Obiecte* (eng. *objects*)
  - Pot primi nume în diagramele de obiecte (pentru a ușura referirea lor), sau pot fi *anonime* (se precizează doar clasa corespunzătoare)
  - Convenție: numele obiectelor se scriu cu litere mici
  - Se precizează valorile atributelor aferente *sloturilor* obiectelor
- Ex.: diagramă de obiecte (eng. *object diagram*) ilustrând obiectele implicate în scenariul *IncendiereDepozit* și legăturile între acestea



# Asocieri și legături

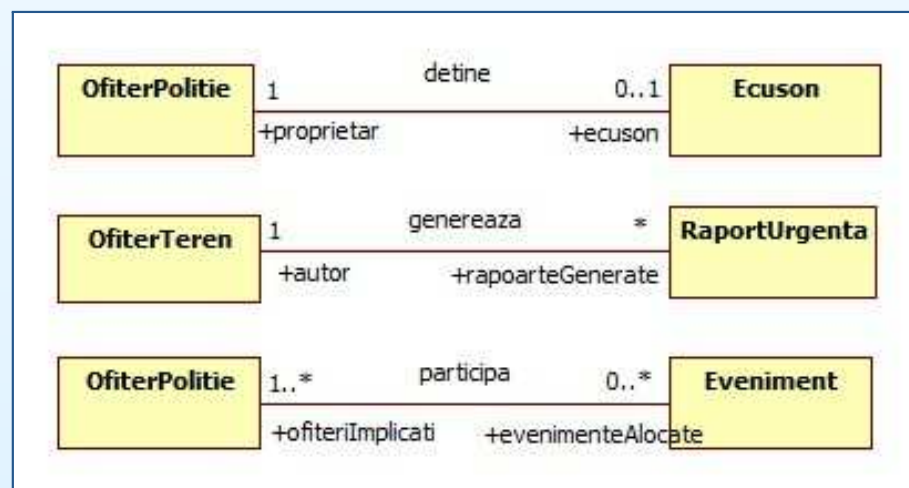
- O *legătură* (eng. *link*) reprezintă o conexiune între două obiecte
- *Asocierile* (eng. *associations*) sunt relații între clase și reprezintă mulțimi de legături
  - Asocierile pot fi uni/bi-direcționale sau cu navigabilitate nespecificată
    - Asocierile unidirecționale indică navigabilitate într-un singur sens - cel precizat de săgeată
    - Asocierile bidirecționale indică navigabilitate în ambele sensuri - se pot reprezenta ambele săgeți, însă, de obicei, se omit
- Ex.: Asociere unidirecțională între clasele *Poligon* și *Punct*



- Conform săgeții, se permite doar navigarea dinspre poligon înspre punct
  - Dat fiind un poligon, se poate interoga colecția punctelor ce definesc acel poligon
  - Dat fiind un punct, NU se pot afla (direct) poligoanele din care face parte

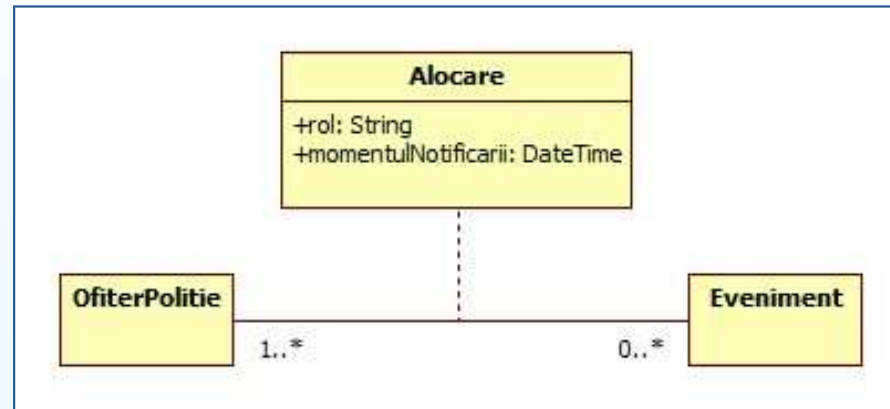
# Multiplicități și roluri

- Fiecare dintre capetele unei asocieri poate fi etichetat cu un *nume de rol* (eng. *role*)
  - Rolurile clarifică scopul asocierii și permit diferențierea între asocieri diferite ce conectează aceleași clase
- Fiecare capăt al unei asocieri poate fi etichetat de o mulțime de întregi = *multiplicitatea* capătului respectiv (eng. *multiplicity*)
  - Multiplicitatea unui capăt de asociere indică numărul de legături pe care o instanță a clasei de la capătul opus îl poate avea cu instanțe ale clasei de la acel capăt
  - Tipuri uzuale de asocieri
    - eng. *one-to-one*  
*one* = 1 or 0..1
    - eng. *one-to-many*  
*many* = 1..\*, 0..\* or \*
    - *many-to-many*

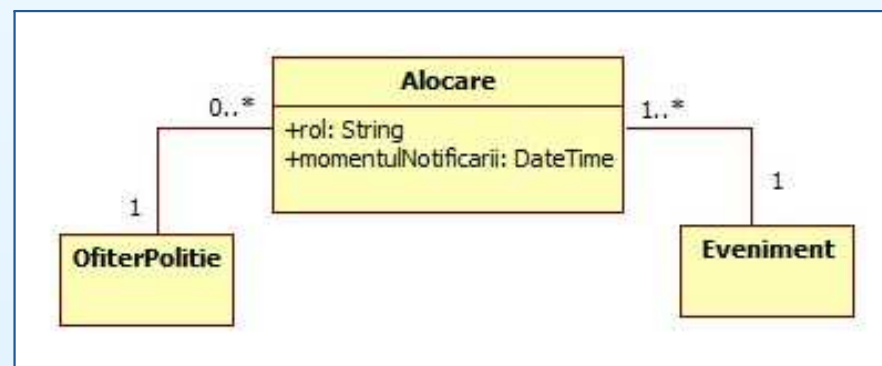


# Clase asociere

- Similar claselor, asocierilor li se pot atașa attribute și operații => *clase asociere* (eng. *association classes*)
  - Sintaxă



- Clasă asociere  $\Leftrightarrow$  clasă + asocieri simple + constrângere de unicitate !

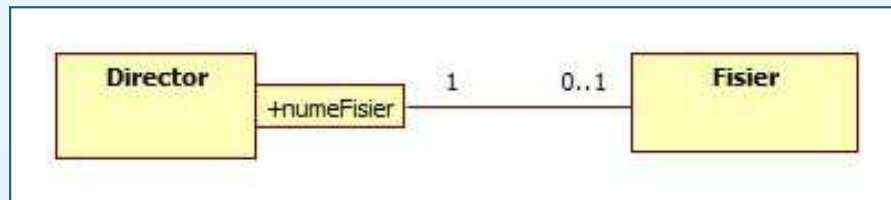


# Asocieri calificate

- *Calificarea* (eng. *qualification*) este o metodă de reducere a multiplicităților, prin utilizarea cheilor (= attribute ce oferă identificare unică)
  - Utilizarea asocierilor calificate simplifică și mărește claritatea diagramelor
- Ex.:
  - Inițial: un director conține mai multe fișiere, fiecare fișier fiind identificat în mod unic în cadrul directorului prin numele său



- Reducerea multiplicității asocierii (introducerea unei asocieri calificate) prin alegerea atributului aferent numelui de fișier ca și calificator (cheie)



# Agregare

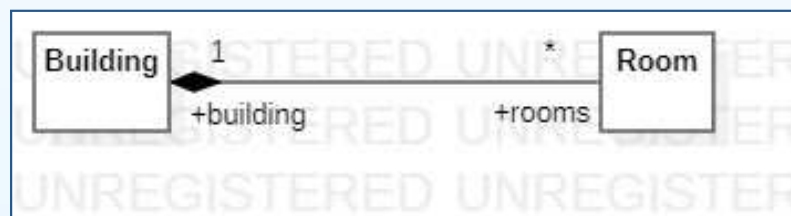
- Reprezintă un caz particular de asociere, care denotă o relație de tip parte - întreg
  - Se traduce prin: *are* (eng. *has*), *constă din*, *este format din*
  - Sintactic, se reprezintă folosind un romb, pe capătul dinspre întreg
- Variante
  - *agregare partajată* (eng. *shared aggregation*)
    - apartenență *slabă* a părților la întreg, părțile pot exista și independent
    - multiplicitatea capătului dinspre întreg poate fi mai mare decât 1 (partea poate aparține mai multor întregi simultan)
    - sintactic, romb gol (engl. *hollow diamond*)





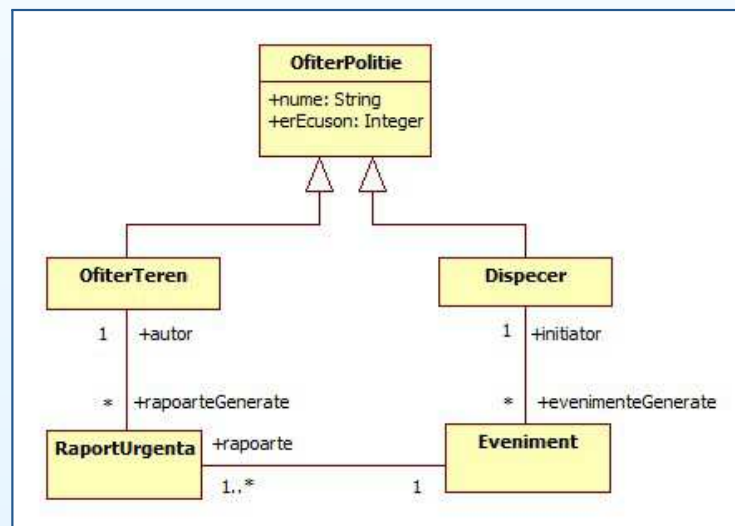
## Agregare (cont.)

- *compunere* (eng. *composition*)
  - o parte poate fi conținută în cel mult un întreg la un moment dat (multiplicitatea maximă pe capătul dinspre întreg e 1)
  - întregul controlează durata de viață a părților (distrugere întregului rezultă în distrugerea părților)
  - sintactic, romb plin (engl. *solid diamond*)



# Generalizare / specializare

- *Generalizarea* este o relație care permite factorizarea atributelor și operațiilor (stării și comportamentului) comune unei mulțimi de clase
  - Clasele *derivate* sau *subclasele* moștenesc attributele și operațiile clasei de bază sau *superclasei*



- UML distinge între conceptul de *operație* și cel de *metodă*
  - operație - specificarea comportamentului
  - metodă - implementarea comportamentului

# Utilizarea diagramelor de clase

---

- În etapa de *analiză a cerințelor*
  - Permit formalizarea cunoștințelor legate de domeniul problemei
  - Clasele corespund entităților din domeniul problemei, iar asocierile relațiilor stabilite între acestea
  - Stabilirea tipurilor atributelor și a signaturilor operațiilor poate fi amânată pentru etapa de proiectare, la fel și deciziile privind navigabilitatea
- În *proiectarea de sistem și obiectuală*
  - Diagramele de clase din analiză sunt rafinate, prin introducerea claselor corespunzătoare entităților din domeniul soluției
  - Clasele sunt grupate în subsisteme cu interfețe bine definite

# Diagrame de interacțiune

---

- *Diagramele de interacțiune* (eng. *interaction diagrams*) descriu șabloane de comunicare într-o mulțime de obiecte care interacționează
  - Obiectele interacționează între ele prin schimb de *mesaje*
  - Recepționarea unui mesaj de către un obiect declanșează execuția unei *metode* a obiectului în cauză, fapt ce determină, de obicei, trimiterea unor mesaje către alte obiecte
  - Un mesaj trimis unui obiect poate avea *argumente* asociate - acestea corespund parametrilor metodei aferente a obiectului destinație
- Tipuri de diagrame de interacțiune
  - Diagrame de secvență (eng. *sequence diagrams*)
  - Diagrame de comunicare (eng. *communication diagrams*)
  - Cele două tipuri de diagrame de interacțiune sunt echivalente: dată fiind o diagramă de secvență, se poate construi diagrama de colaborare echivalentă și reciproc

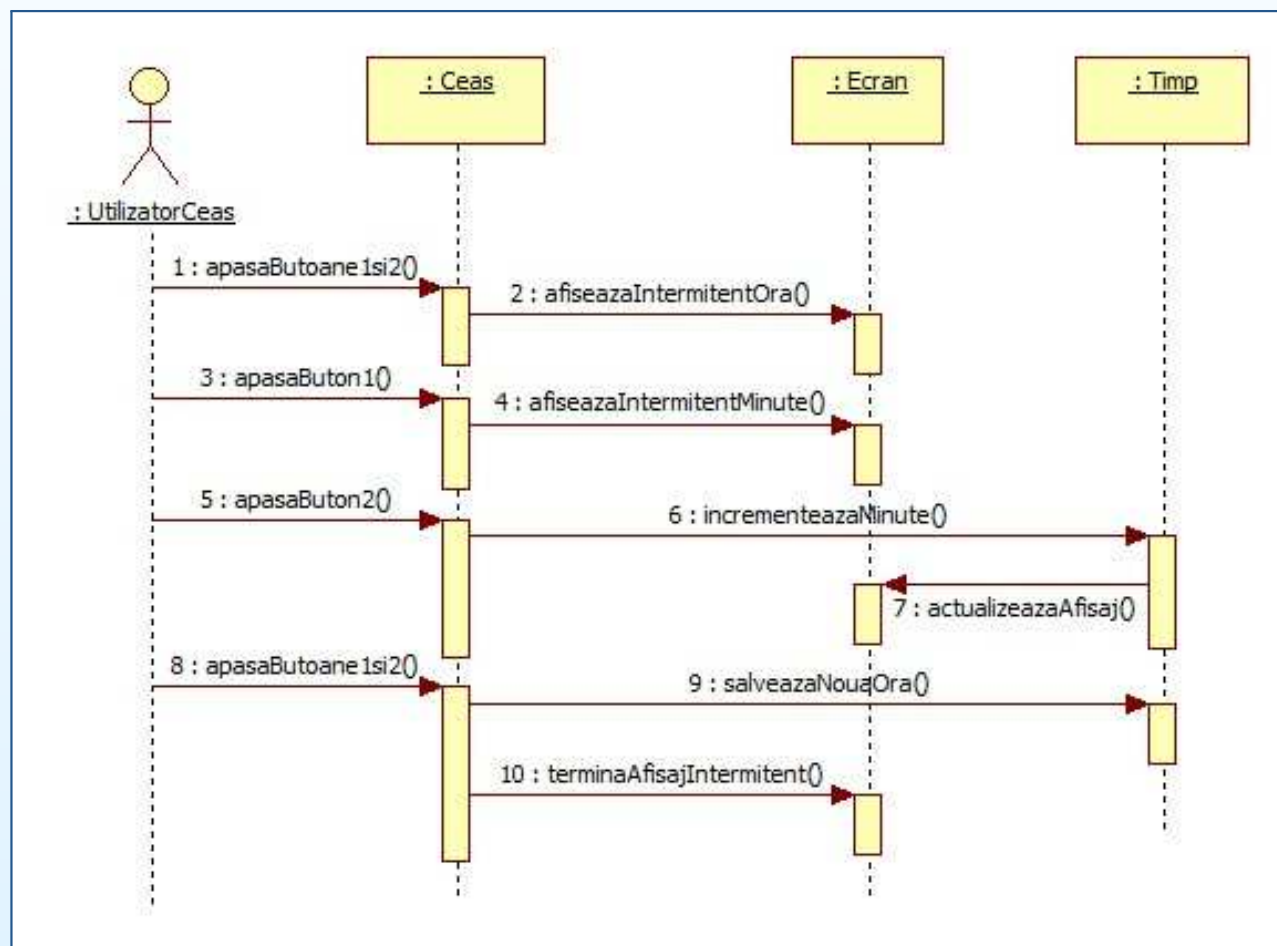
## Exemplu: ceas electronic

---

- Considerăm exemplul unui ceas electronic simplu, cu două butoane. Pentru a modifica timpul curent, utilizatorul trebuie să apese simultan cele două butoane, moment în care se intră în modul de lucru *Setare timp*. În acest mod de lucru, ceasul afișează intermitent componenta modificată (oră, minut, secundă, zi, lună sau an). Imediat după intrarea în modul *Setare timp*, va fi afișată intermitent ora. În cazul în care utilizatorul apasă primul buton, va fi afișată intermitent următoarea componentă. Dacă este apăsat al doilea buton, componenta curentă afișată intermitent va fi incrementată cu o unitate (atingerea valorii finale a intervalului aferent determină resetarea la valoarea inițială a componentei respective). Modul *Setare timp* poate fi părăsit prin apăsarea simultană a ambelor butoane.

## Diagrame de secvență

- Ex.: diagramă de secvență corespunzătoare incrementării minutelor curente ale unui ceas electronic cu o unitate (scenariu aferent cazului de utilizare *Schimba ora*)



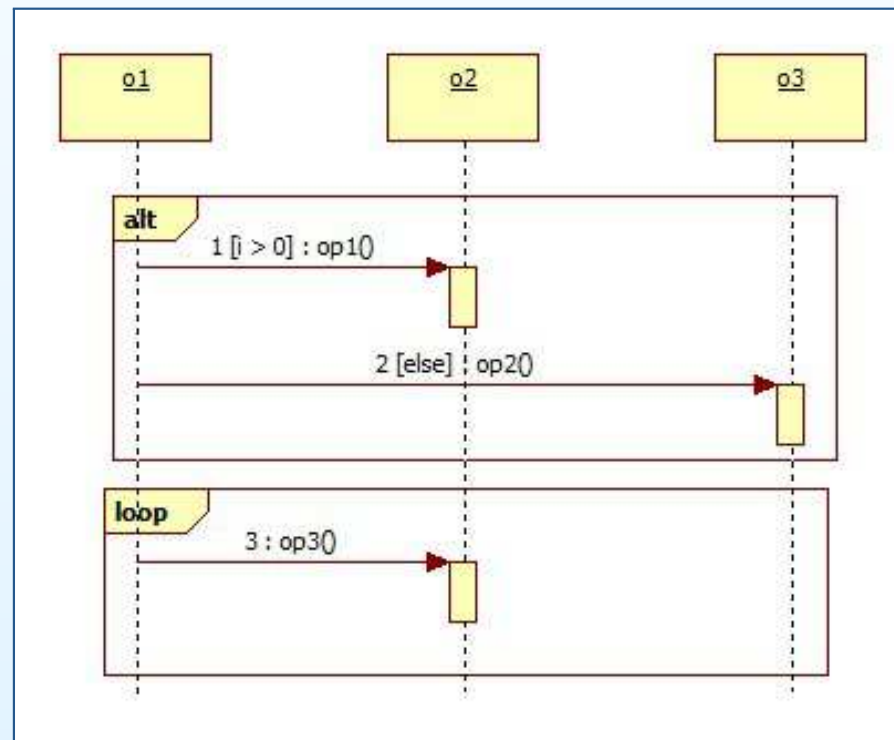
## Diagrame de secvență (cont.)

---

- Primează perspectiva temporală
- Obiectele participante la interacțiune se reprezintă pe orizontală, iar timpul pe verticală
  - Fiecare coloană corespunde unui obiect participant la interacțiune
  - Coloana cea mai din stânga corespunde unei instanțe a actorului care declanșează interacțiunea
- Transmiterea mesajelor este reprezentată prin săgeți etichetate
  - Etichetele indică numele mesajelor și eventualele argumente
- Activarea (execuția unei metode) este reprezentată prin dreptunghiuri verticale
- Mesajele inițiate de către actori corespund unor interacțiuni descrise în cazul de utilizare aferent
  - Deși, pentru simplitate, interacțiunile dintre obiecte și cele între sistem și actori sunt reprezentate uniform prin mesaje, cele două tipuri de interacțiune sunt de natură diferită!

## Diagrame de secvență (cont.)

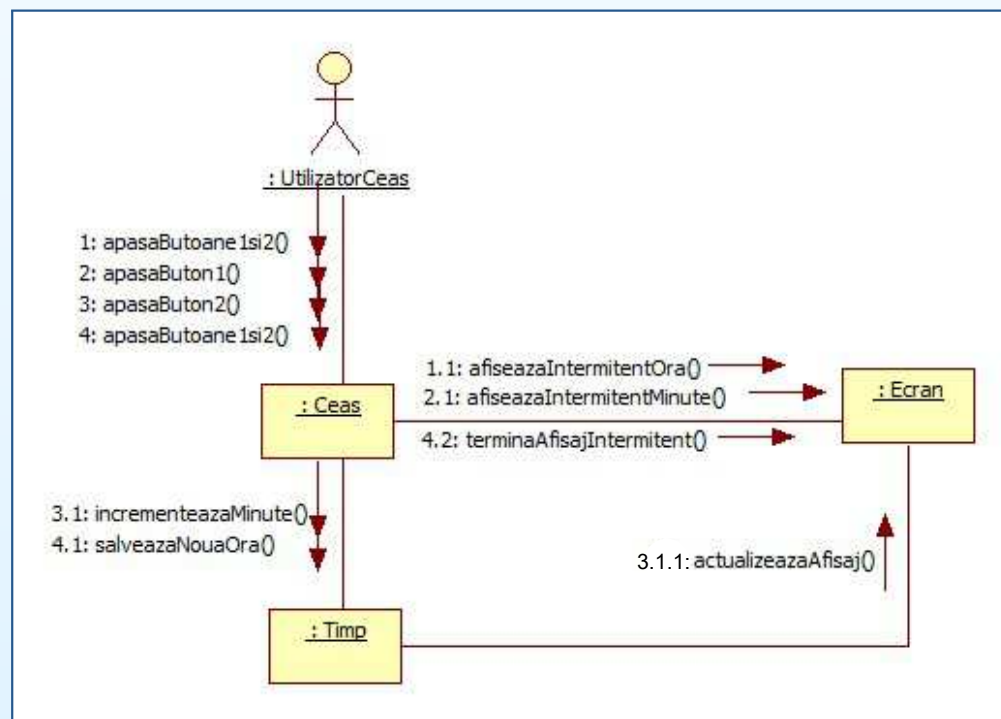
- Diagramele de secvență pot fi utilizate pentru descrierea unor interacțiuni specifice (scenarii) sau a unora generale (cazuri de utilizare)
  - Corespunzător celei de-a doua situații, există notații specifice pentru condiționări și cicluri (eng. *combined fragments*)





# Diagrame de comunicare

- Ilustrează aceeași informație ca și diagramele de secvență, însă cu accent pe colaborările între obiectele participante la interacțiune (în diagramele de secvență accentul e pe secvențierea mesajelor în timp)
  - Avantaje: aspect compact al diagramei
  - Dezavantaje: secvențierea mesajelor e dificil de urmărit
- Ex.: diagramă de comunicare corespunzătoare incrementării minuteror curente ale unui ceas electronic cu o unitate



# Utilizarea diagramelor de interacțiune

---

- Motivul principal al construirii digramelor de interacțiune îl reprezintă identificarea responsabilităților claselor existente deja în diagrama de clase, precum și identificarea de noi clase
- În mod uzual, se realizează cel puțin câte o diagramă de interacțiune pentru fiecare caz de utilizare, axată pe fluxul normal de evenimente + diagrame aferente fluxurilor de excepție
  - Sunt identificate obiectele care participă la cazul respectiv de utilizare și se atribuie fragmente din comportamentul ce definește cazul de utilizare acestor obiecte, sub forma operațiilor
- După definirea diagramei inițiale de clase, aceasta și diagramele de interacțiune se dezvoltă în tandem
  - Acest proces are ca și efect rafinarea cazurilor de utilizare (rezolvarea ambiguităților, adăugarea unor noi elemente de comportament), cu introducerea de noi obiecte și servicii

# Diagrame de tranziție a stărilor

---

- O *mașină cu stări UML* (eng. *UML state machine*) reprezintă o notăție folosită pentru a descrie succesiunea de stări prin care trece un obiect sub acțiunea evenimentelor externe
- Originea *diagramelor de tranziție a stărilor UML* (eng. *UML state machine diagrams*) este reprezentată de teoria automatelor finite, extinsă cu
  - o notăție pentru imbricarea stărilor și a mașinilor cu stări (o stare poate fi descrisă prin intermediul unei mașini cu stări)
  - o notăție ce permite etichetarea tranzițiilor cu mesaje trimise și condiții impuse asupra obiectelor
- Mașinile cu stări UML sunt bazate în principal pe diagramele de stări (eng. *statecharts*) introduse de Harel [Harel, 1987]

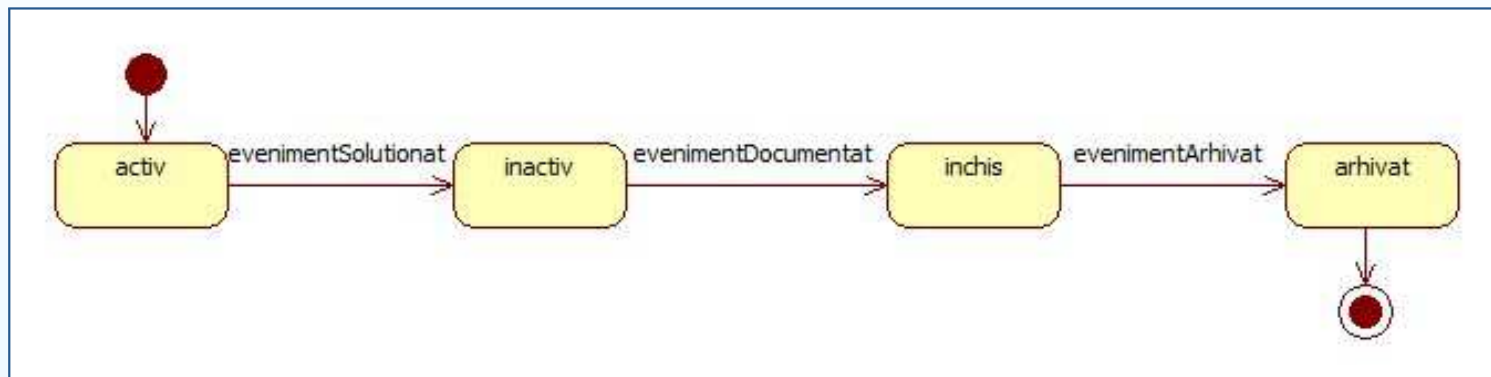
# Stări și tranziții

---

- O *stare* (eng. *state*) reprezintă o condiție satisfăcută de valorile atributelor unui obiect
  - Ex.: un obiect de tip *Eveniment* din sistemul SGA poate fi în una din patru stări posibile:
    - *activ* - denotă o situație care necesită a fi soluționată (ex.: un incendiu, un accident rutier)
    - *inactiv* - denotă o situație care a fost rezolvată, dar nu a fost încă documentată corespunzător (ex.: incendiul a fost stins, dar nu au fost estimate încă pagubele)
    - *închis* - denotă o situație care a fost rezolvată și documentată corespunzător
    - *arhivat* - denotă un eveniment închis a cărui documentație a fost arhivată
  - Aceste patru stări pot fi reprezentate adăugând în clasa *Eveniment* un atribut *status*, ce poate avea, la un moment dat, una din aceste patru valori: *activ*, *inactiv*, *închis*, *arhivat*
- O *tranziție* (eng. *transition*) reprezintă o schimbare de stare, care poate fi provocată de declanșarea unor evenimente, îndeplinirea unor condiții sau de trecerea unui anumit interval de timp

## Stări și tranziții (cont.)

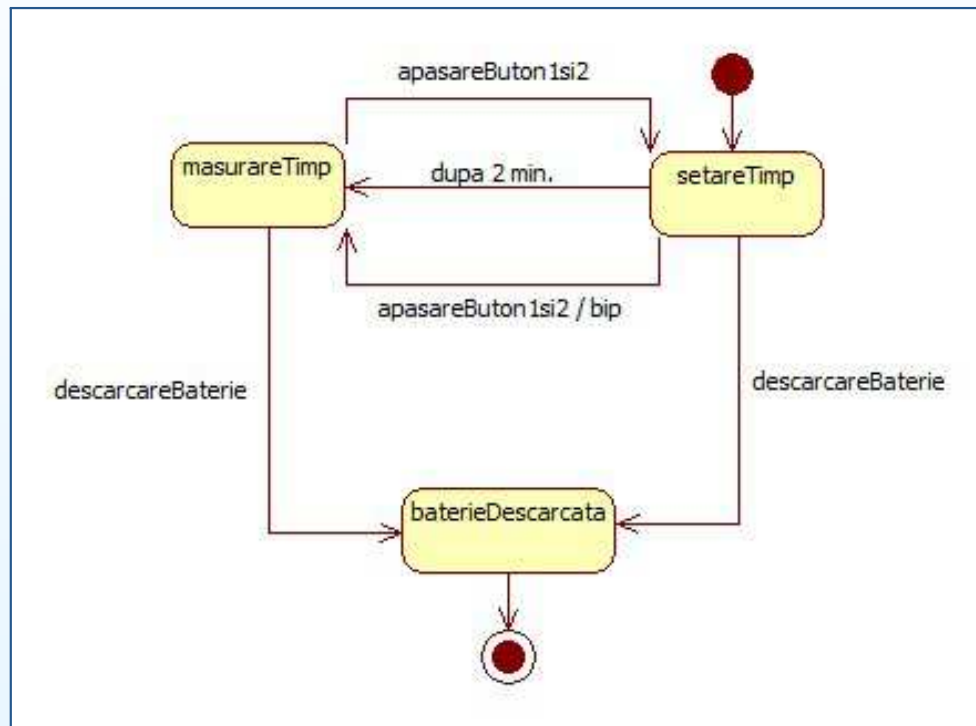
- Ex.: diagramă de tranziție a stărilor UML aferentă clasei *Eveniment*



- Sintactic:
  - O stare se reprezintă ca un dreptunghi cu colțuri rotunjite
  - O tranziție se reprezintă ca o săgeată ce unește două stări
  - Stările sunt etichetate cu numele lor
  - Tranzițiile pot fi etichetate cu numele evenimentelor care le declanșează
  - Un disc denotă (pseudo)starea inițială
  - Un disc încercuit denotă o (pseudo)stare finală

## Stări și tranziții (cont.)

- Ex.: diagramă de tranziție a stărilor UML aferentă clasei *Ceas*



- Tranziția de la starea *setareTimp* la starea *măsurareTimp* poate fi declanșată de un eveniment (*apasareButon1si2*) sau de trecerea timpului (2 min.)
- Tranziția declanșată de eveniment de la starea *setareTimp* la starea *măsurareTimp* are asociată o acțiune (semnal sonor *bip*)

# Acțiuni, tranziții interne și activități

---

- O *acțiune* (eng. *action*) reprezintă o unitate fundamentală de procesare, care poate primi input-uri, poate produce output-uri și poate schimba starea sistemului
  - Acțiunile sunt considerate atomice (se execută într-un timp scurt și nu pot fi întrerupte)
  - Ex. de acțiune: apelul unei operații
  - Într-o mașină cu stări, acțiunile pot fi localizate:
    - la nivelul unei tranziții (ex.: acțiunea *bip*, asociată tranziției de la starea *setareTimp* la starea *măsurareTimp*)
    - la intrarea într-o stare, introduse prin eticheta *entry* (ex.: acțiunea *afiseaza intermitent ore*, la intrarea în starea *setareTimp*)
    - la ieșirea dintr-o stare, introduse prin eticheta *exit* (ex. acțiunea *termina afisaj intermitent*, la ieșire din starea *setareTimp*)
  - În timpul unei tranziții, se execută mai întâi acțiunile de ieșire din starea sursă, apoi acțiunile asociate tranziției, iar la final acțiunile de intrarea ale stării destinație
  - Acțiunile de intrare/ieșire se execută ori de câte ori se intră în / iese din starea respectivă, indiferent de tranziția implicată în schimbarea de stare

## Acțiuni, tranziții interne și activități (cont.)

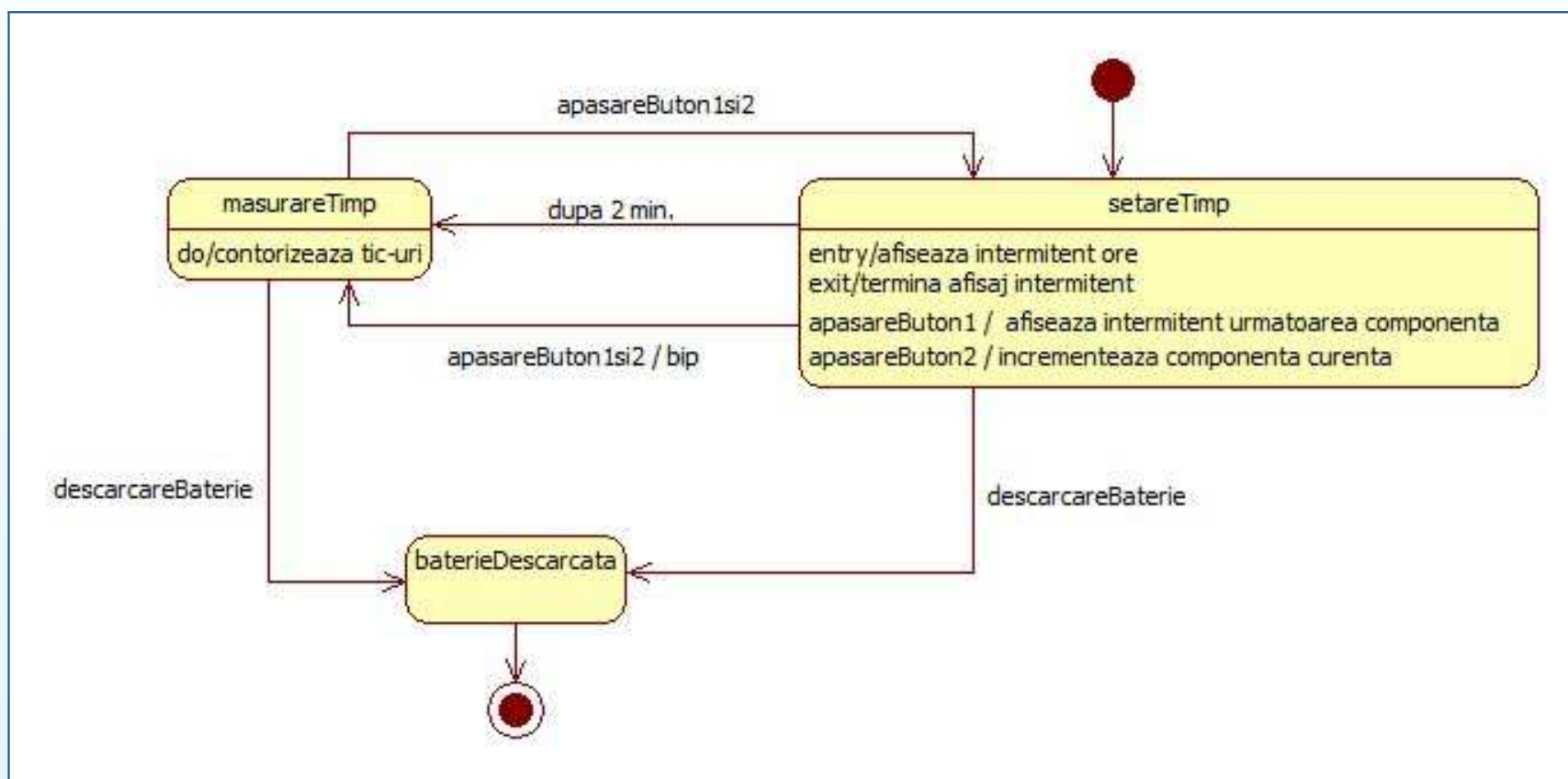
---

- O *tranziție internă* (eng. *internal transition*) este o tranziție ce nu determină părăsirea stării curente
  - Tranzițiile interne sunt declanșate de evenimente și pot avea acțiuni asociate
    - Ex.: tranzițiile interne din starea *setareTimp*, declanșate de evenimentele *apasareButon1* și *apasareButon2*
  - Declanșarea unei tranziții interne nu determină execuția acțiunilor de intrare/ieșire din stare
- O *activitate* (eng. *activity*) reprezintă o mulțime coordonată de acțiuni
  - O stare poate avea o activitate asociată, care se execută atâta timp cât obiectul rămâne în acea stare
  - Spre deosebire de acțiuni, care sunt atomice, activitățile durează mai mult timp și pot fi întrerupte de ieșirea obiectului din starea curentă
  - Activitățile sunt introduse prin eticheta */do* și plasate în starea în care se execută
    - Ex.: activitatea *contorizează tic-uri* din starea *masurareTimp*



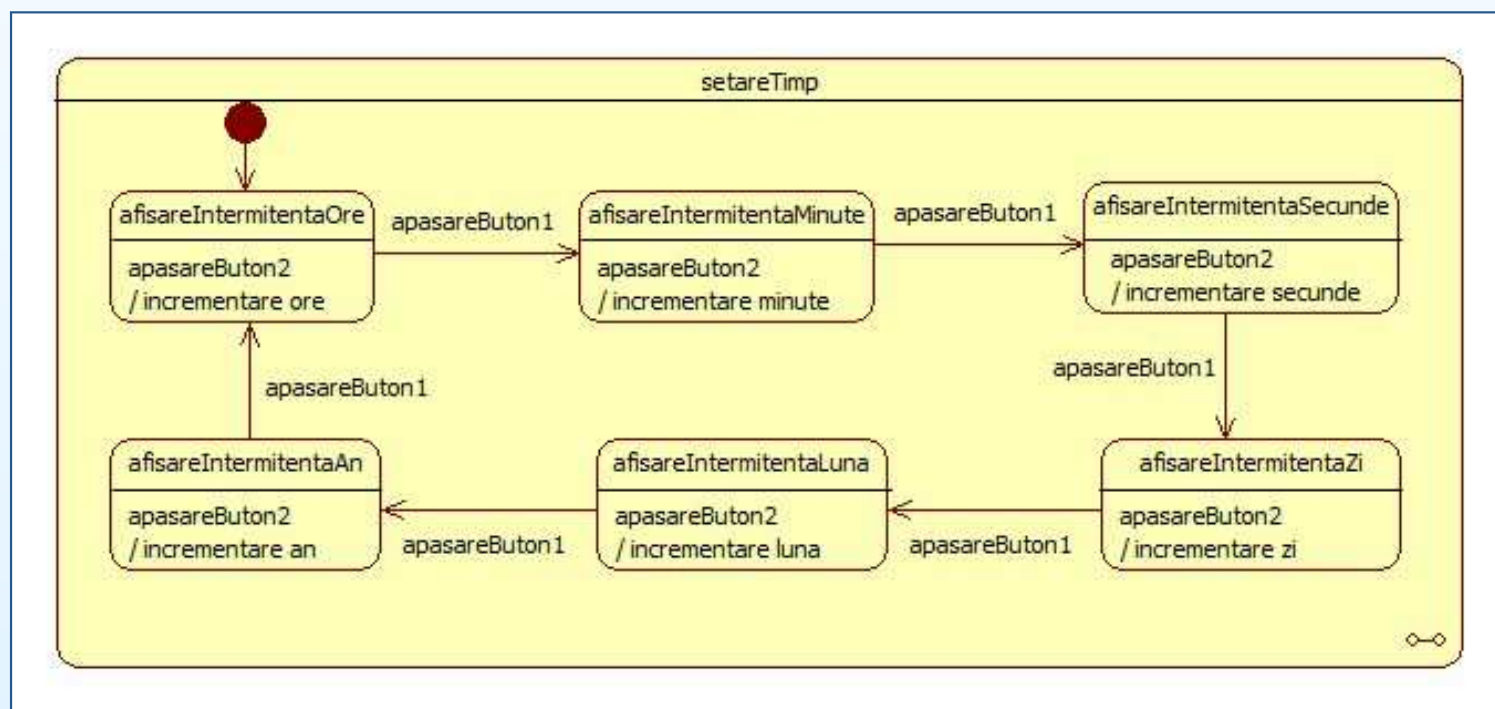
## Acțiuni, tranziții interne și activități (cont.)

- Ex.: Rafinare a diagramei de tranziție a stărilor aferente clasei Ceas, cu ilustrarea unor acțiuni, tranziții interne și activități



# Imbricarea mașinilor cu stări

- Reprezintă o alternativă la folosirea tranzițiilor interne
- Crește inteligibilitatea și permite reducerea complexității diagramelor
- Ex.: Rafinare a stării *setareTimp* prin eliminarea/relocarea tranzițiilor interne și imbricarea unei submașini cu stări



- Fiecare dintre tranzițiile interne ale substărilor ar putea fi reprezentată, într-o rafinare ulterioară, ca o submașină cu stări

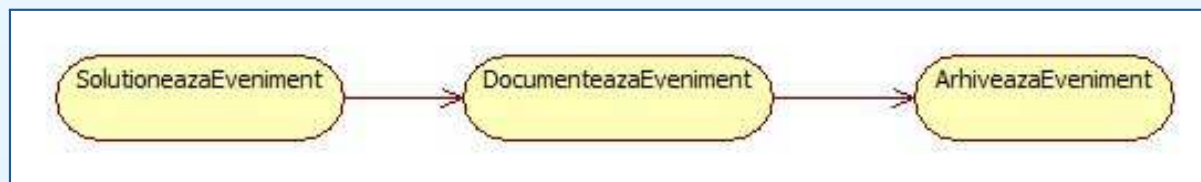
## Utilizarea diagramelor de tranziție a stărilor

---

- Diagramele de tranziție a stărilor sunt folosite pentru modelarea comportamentului netrivial al obiectelor sau subsistemelor individuale
- În etapa de analiză, utilizarea lor ajută la identificarea atributelor obiectelor din domeniul problemei și la rafinarea descrierii comportamentului acestora
- În etapa de proiectare, pot fi folosite pentru descrierea obiectelor din domeniul soluției care prezintă comportament complex, dependent de stare (reacționează diferit la același stimul, funcție de starea în care se află)
  - Șablonul de proiectare *State*

# Diagrame de activități

- O *diagramă* de activități (eng. *activity diagram*) descrie modul de realizare a unui anumit comportament în termenii uneia sau a mai multor secvențe de activități și a fluxurilor de obiecte necesare pentru coordonarea acestor activități
  - Diagramele de activități sunt ierarhice: o activitate reprezintă fie o *acțiune*, fie un *graf de subactivități* cu fluxurile de obiecte aferente
- Ex.: diagramă de activități aferentă gestionării unui *Eveniment* din sistemul SGA
  - Activitățile sunt reprezentate prin dreptunghiuri rotunjite
  - Săgețile dintre activități reprezintă fluxul de control
  - Execuția activităților este secvențială: o activitate se poate executa doar după terminarea activităților care o preced



- *SolutioneazaEveniment* - dispecerul primește rapoarte și alocă resurse
- *DocumenteazaEveniment* - ofițerii și dispecerii implicați documentează evenimentul

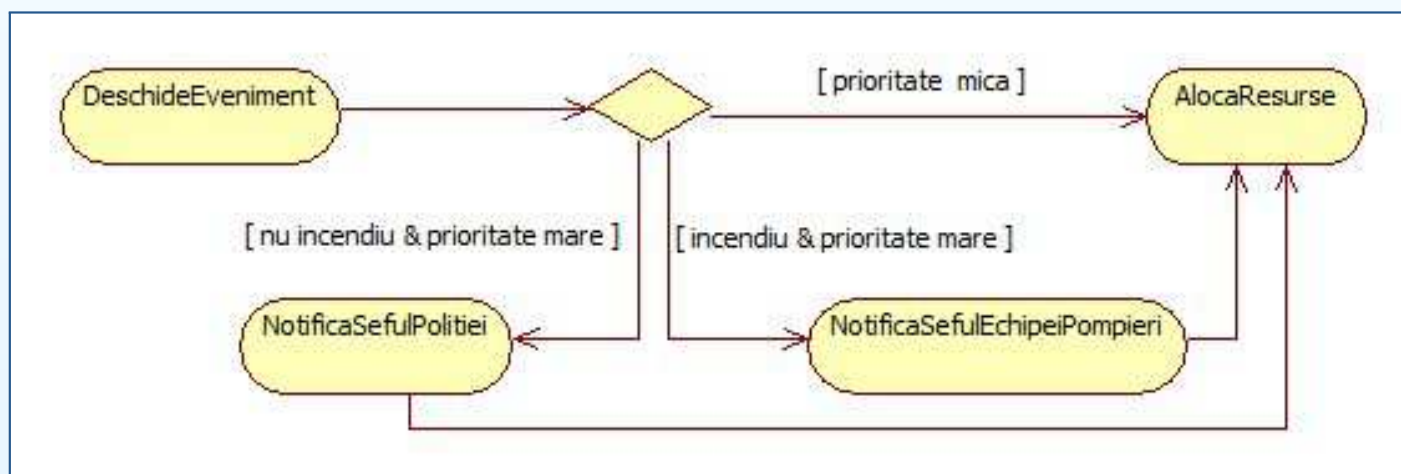
# Elemente de control

---

- *Elementele de control* (eng. *control nodes*) permit coordonarea fluxului de control dintr-o diagramă de activități, oferind mecanisme de reprezentare a deciziilor, a concurenței și sincronizării
- Tipuri principale de elemente de control
  - noduri decizionale
  - noduri fork
  - noduri join

## Noduri decizionale

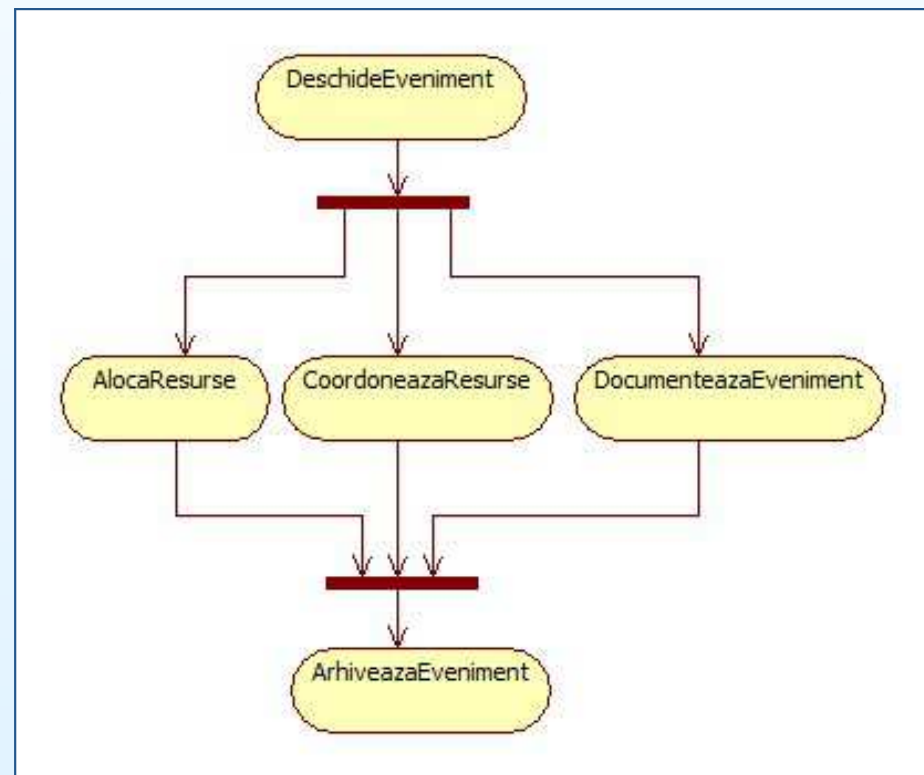
- *Nodul decizional* (eng. *decision node*) reprezintă o ramificare a fluxului de control, ce denotă alternative pe baza unei condiții relativ la starea unui obiect sau grup de obiecte
- Ex.: Nod decizional pentru gestionarea evenimentelor funcție de prioritate și tip



- Nodurile decizionale se reprezintă folosind un romb cu una sau mai multe săgeți de intrare și două sau mai multe săgeți de ieșire
- Ramurile de ieșire sunt etichetate cu condițiile aferente selectării lor în fluxul de control

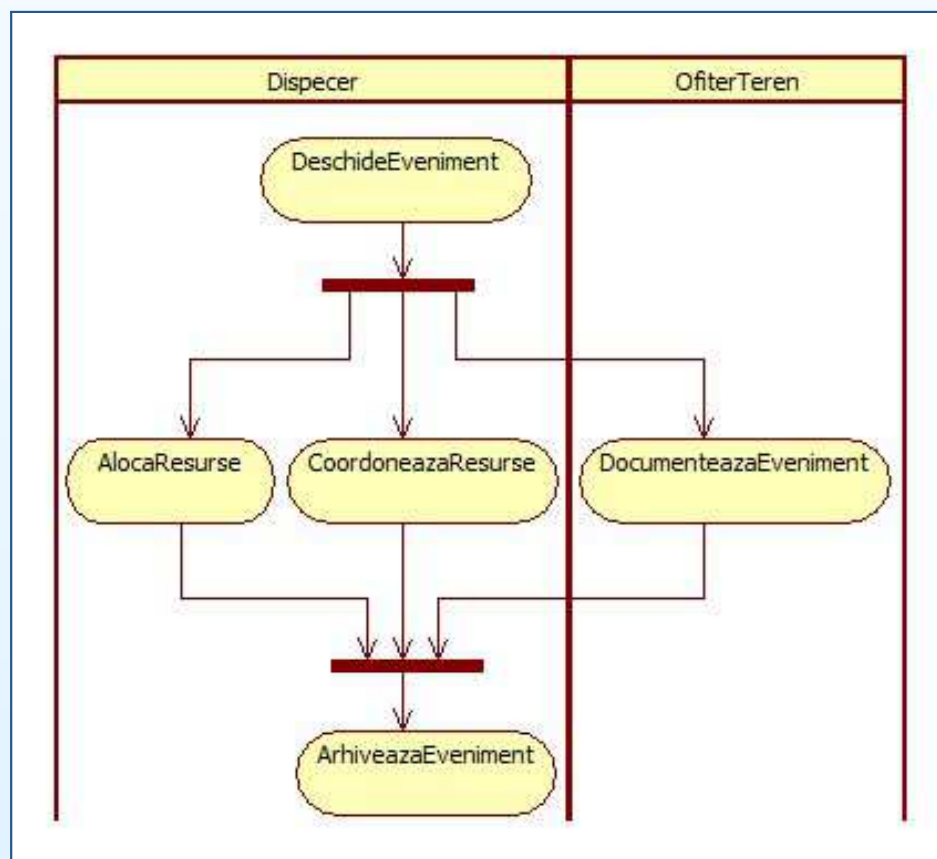
## Noduri fork și join

- *Nodurile fork și join* permit reprezentarea concurenței și a sincronizării
  - Nodurile fork indică divizarea fluxului de control în thread-uri
  - Nodurile join indică sincronizarea thread-urilor și combinarea fluxurilor de control într-un singur thread
- Ex.: Activitățile *AlocaResurse*, *CoordoneazaResurse*, *DocumenteazaEveniment* pot fi efectuate în paralel, însă doar după terminarea activității *DeschideEveniment*. Activitatea *ArhiveazaEveniment* nu poate fi inițiată decât după terminarea tuturor celor trei activități concurente.



## Partiționarea activităților

- Activitățile pot fi grupate pe *partiții* (eng. *swimlanes*), pentru a indica obiectul sau subsistemul care le va implementa
  - Partițiile sunt reprezentate ca și dreptunghiuri ce conțin grupuri de activități
  - Tranzițiile pot intersecta partițiile
- Ex.: Partiționarea activităților legate de gestiunea evenimentelor





# Utilizarea diagramelor de activități

---

- Diagramele de activități oferă o vedere centrată pe sarcini a comportamentului unei mulțimi de obiecte
- Pot fi utilizate pentru
  - Descrierea constrângerilor privind secvențierea cazurilor de utilizare
  - Descrierea activităților secvențiale în cadrul unui grup de obiecte
  - Descrierea sarcinilor unui proiect

## Referințe

---

- [Rumbaugh et al., 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.
- [Booch, 1994] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994 .
- [Jacobson et al., 1992] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley, Reading, MA, 1992.
- [Harel, 1987] D. Harel, *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, pp. 231-274, 1987.