Fundamente de Inginerie Software

Cap. 4

Limbaje de Modelare Software.

Conf.Dr.Ing. Dan Pescaru

Textbooks: Maciaszek "Practical Software Engineering", 2005, Cap. 2

Sursă: http://www.comp.mq.edu.au/books/pse/

2009

Limbaje de modelare

- Ingineria software se sprijină pe modelare:
  - Produsul fina – programul – este un model executabil
- Modelarea necesită un limbaj adecvat pentru:
  - Comunicarea între parteneri
  - Exprimarea proceselor de dezvoltare şi a artefactelor pe multiple nivele de abstractizare
- Limbajul UML (Unified Modeling Language)
- Profile UML pentru diverse domenii şi tehnologii
- Un model consta în una sau mai multe diagrame precum şi în informațiile adiționale stocate în cadrul proiectului

Limbaje de modelare structurate

- Programarea structurată
  - Fără instrucțiuni *go to*
  - Structuri de control condiționate (*if, switch*) şi ciclice (*for, while*)
  - Stil de proiectare top-down a programelor
- Programare structurată → modelare structurată (analiză şi proiectare structurată)
  - Exprimă caracterul monolitic şi procedural al sistemelor stil Cobol
  - Decompoziția funcțională – dezvoltare top-down orientată funcțional
  - Tehnici de vizualizare
    - Diagrame de flux de date (Data Flow Diagrams DFD)
    - Diagrame Entitate-Relație (Entity-Relatioship Diagrams ERD)
    - Diagrame structurale (Structure Charts SC)

Diagrame de flux de date

- Una dintre tehnicile cele mai răspândite de modelare din istoria ingineriei software
- Astăzi este pusă în umbră de modelele orientate pe obiecte
- Se bazează pe decompoziția funcțională

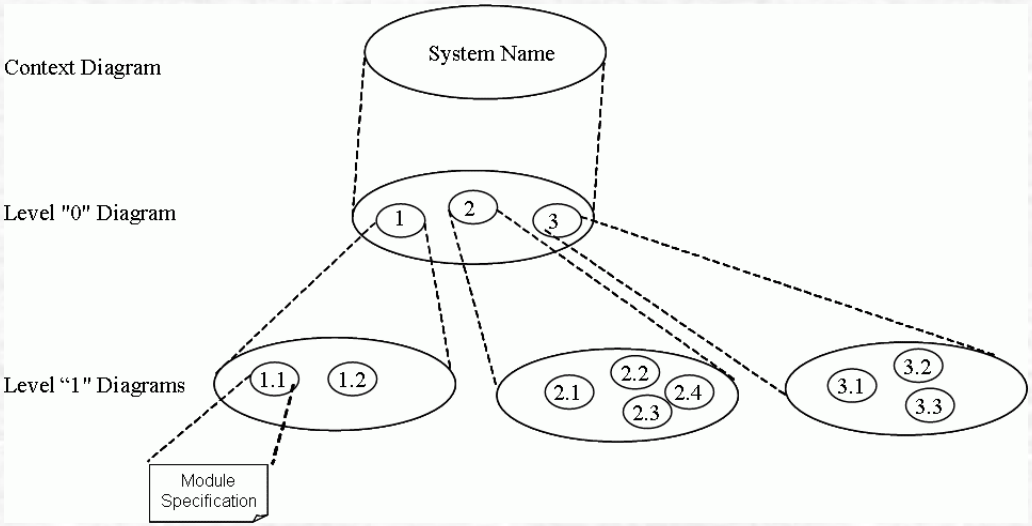
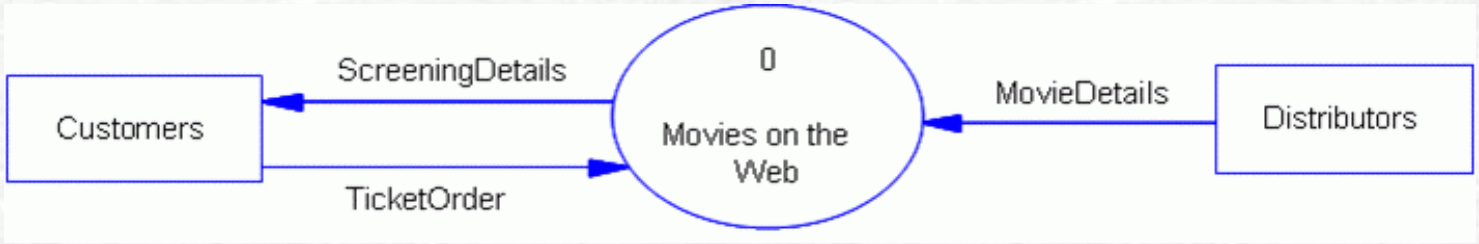


Diagrama de context

- Constă în:
  - Un singur proces
  - Un număr de entități externe
  - Fluxuri de intrare-ieşire între proces şi entitățile externe
- Denotă locul sistemului în mediul în care va evolua, stabilind granițele cu acesta



- *Ex: un sistem de reclamă pe Web a unui lanț de cinematografe (Maciaszek, 2005)*

Diagrama de nivel “0”

- Denumită şi diagramă de ansamblu
- (CRUD = **C**reate, **R**ead, **U**pdate and **D**elete)

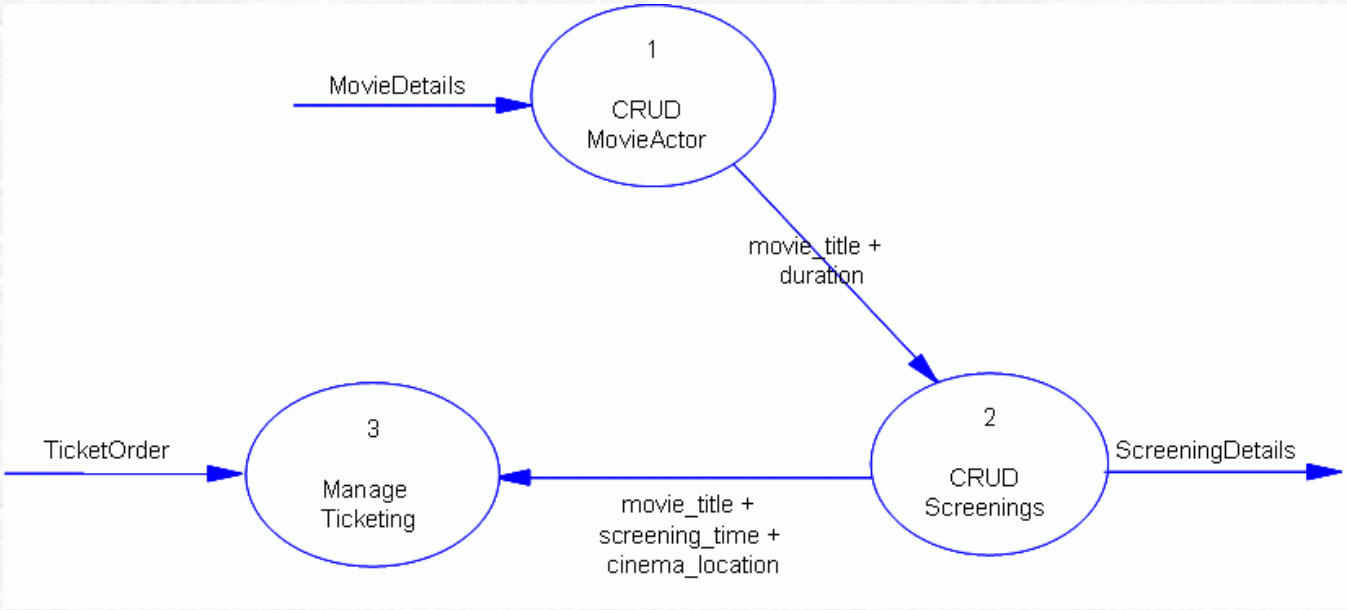
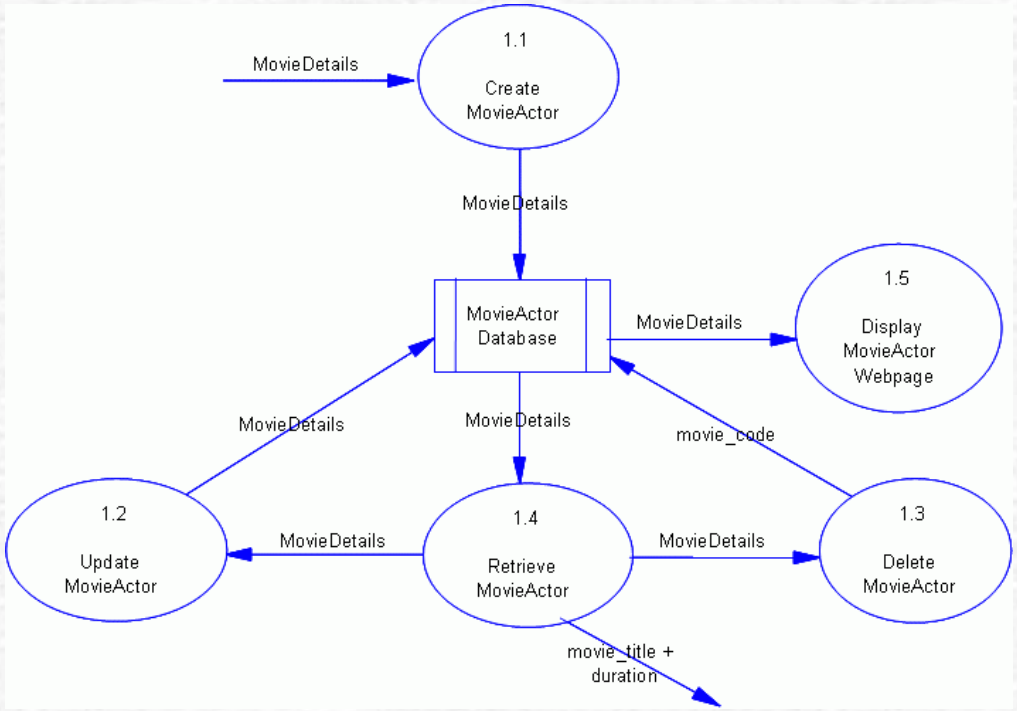


Diagrama de nivel “1”

- Balansarea fluxurilor (între procese), module de stocare a datelor
- Decompoziție a proceselor de pe nivelul “0” (ex. CRUD MovieActor)



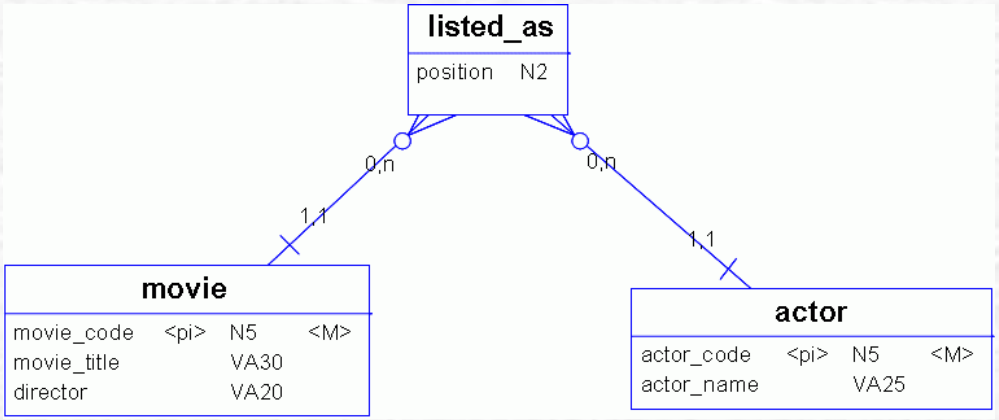
Modele Entitate-Relaţie (ER)

- O tehnică de modelare a datelor
- Diagramele ER pot să conțină doar trei elemente de modelare: entități, relații şi atribute

- O **Entitate** este o structură de date conceptuală reprezentând un fapt sau regulă distinct identificabilă în problemă
- O **Relație** reprezintă o asociere între instanțele entităților
- Un **Atribut** este o pereche data-valoare de tip:
  - Valoare singulară
  - Valoare multiplă (atribut compus) – în general de evitat

ER "Crow's foot notation"

- Atributele apar în cadrul entităților
  - Nume, tip, dacă este cheie (pi – primary identifier), dacă trebuie să aibă o valoare (m – mandatory)
- Multiplicitatea relațiilor
  - Participare în relație opțională sau obligatorie



Limbaje de Modelare Orientate pe Obiecte

- **Unified Modeling Language** (UML) *"…is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems."* *(UML, 2003b, p.1-1)*
- În UML modelarea vizuală constă în principal în aranjarea aşa numiților ***clasificatori***
  - Un **Clasificator** este un element de modelare care descrie comportamentul sau structura sistemului printr-o reprezentare vizuală
    - Exemple de clasificatori: class, actor, use case, relationship
- La rularea sistemului, clasificatorilor le corespund obiectele
  - Un **Obiect** este o bucată de cod care are:
    - Stare – definită de valorile atributelor
    - Comportament – definit de operațiile care le poate executa
    - Identitate – care îl diferențiază de alte obiecte (chiar cu aceiaşi stare şi comportament)

Tipuri de Diagrame UML

- Class Diagram (state structure)
- Use Case Diagram
- Interaction Diagram
- Statechart Diagram
- Activity Diagram
- Implementation Diagram

UML Class Diagram

- Class diagram (diagramă de clase)
  - Exprimă în mod static structura modelelor (numită şi model de stare)
  - Vizualizează clase (şi interfețe), structura lor internă şi relațiile lor cu alte clase (interfețe)
- *"a* **Class** *is the descriptor for a set of objects with similar structure, behavior, and relationships" (UML, 2003, p.3-35)*
- **Atributul** este o caracteristică (tipizată) structurală a unei clase
  - Poate fi: data membru, variabilă membru, variabilă de instanță sau câmp
- **Operația** este o caracteristică comportamentală a unei clase
  - Poate fi: funcție membră sau metodă

Elemente de Modelare la o Diagramă de Clase

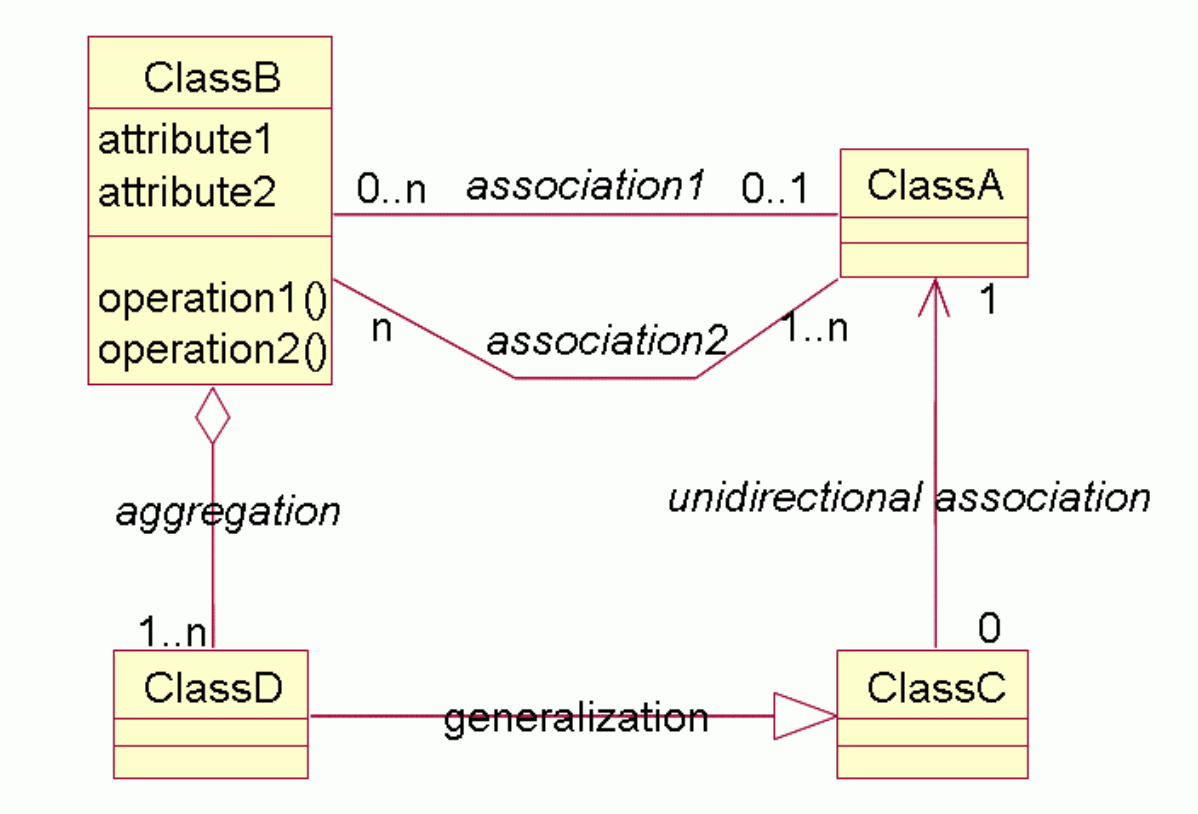
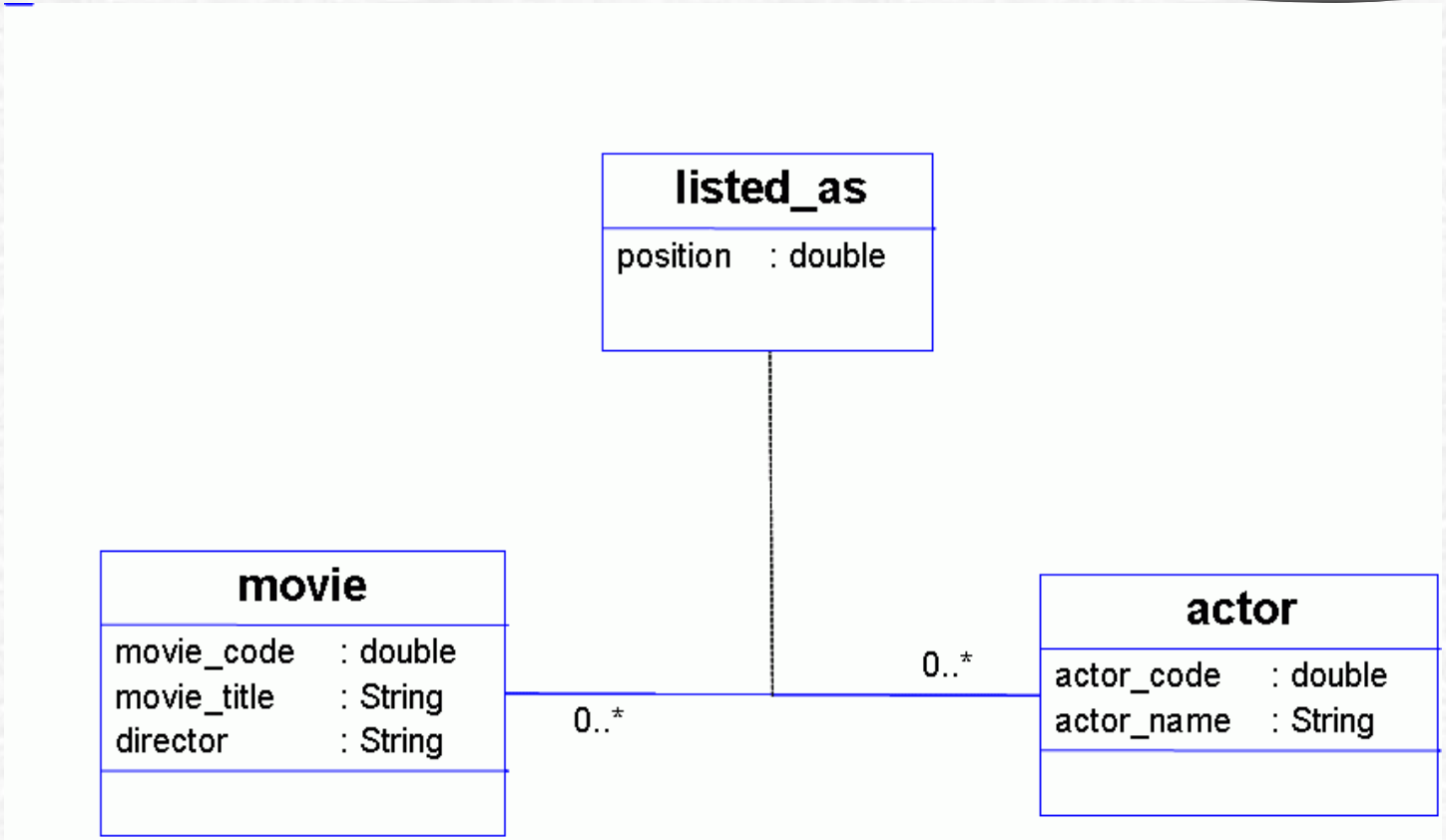
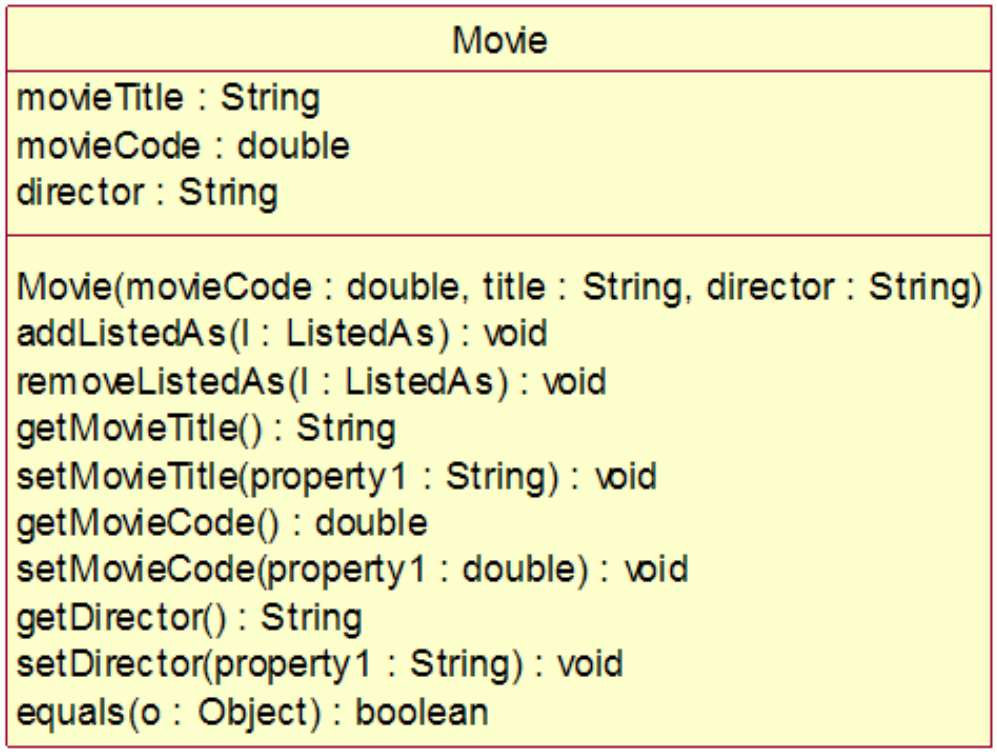


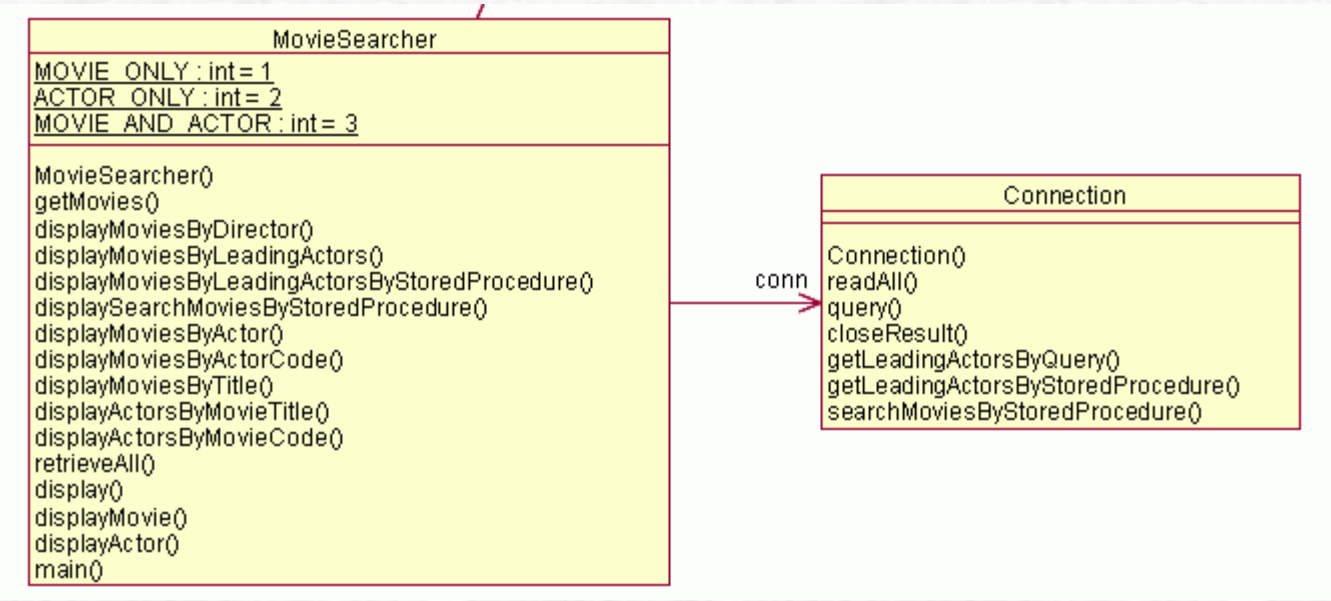
Diagrama de Clase ca o Diagramă de Structură Statică



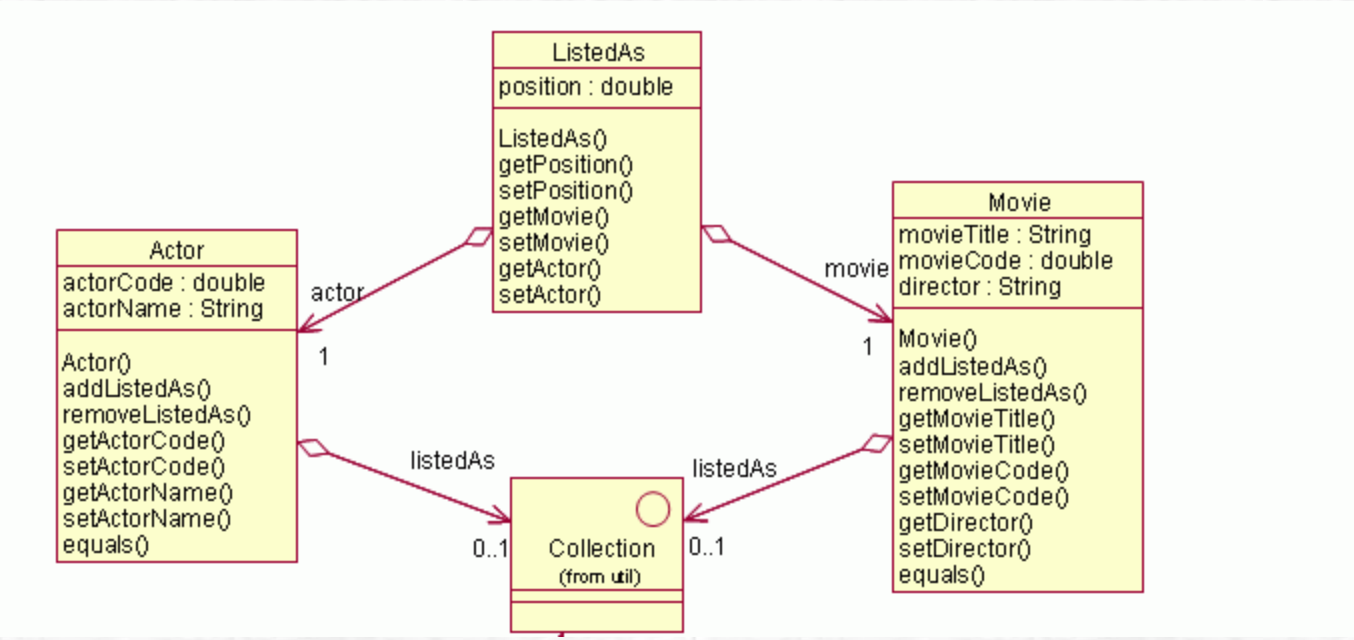
Caracteristici de Stare şi Comportament



Proiectarea Diagramelor de Clase



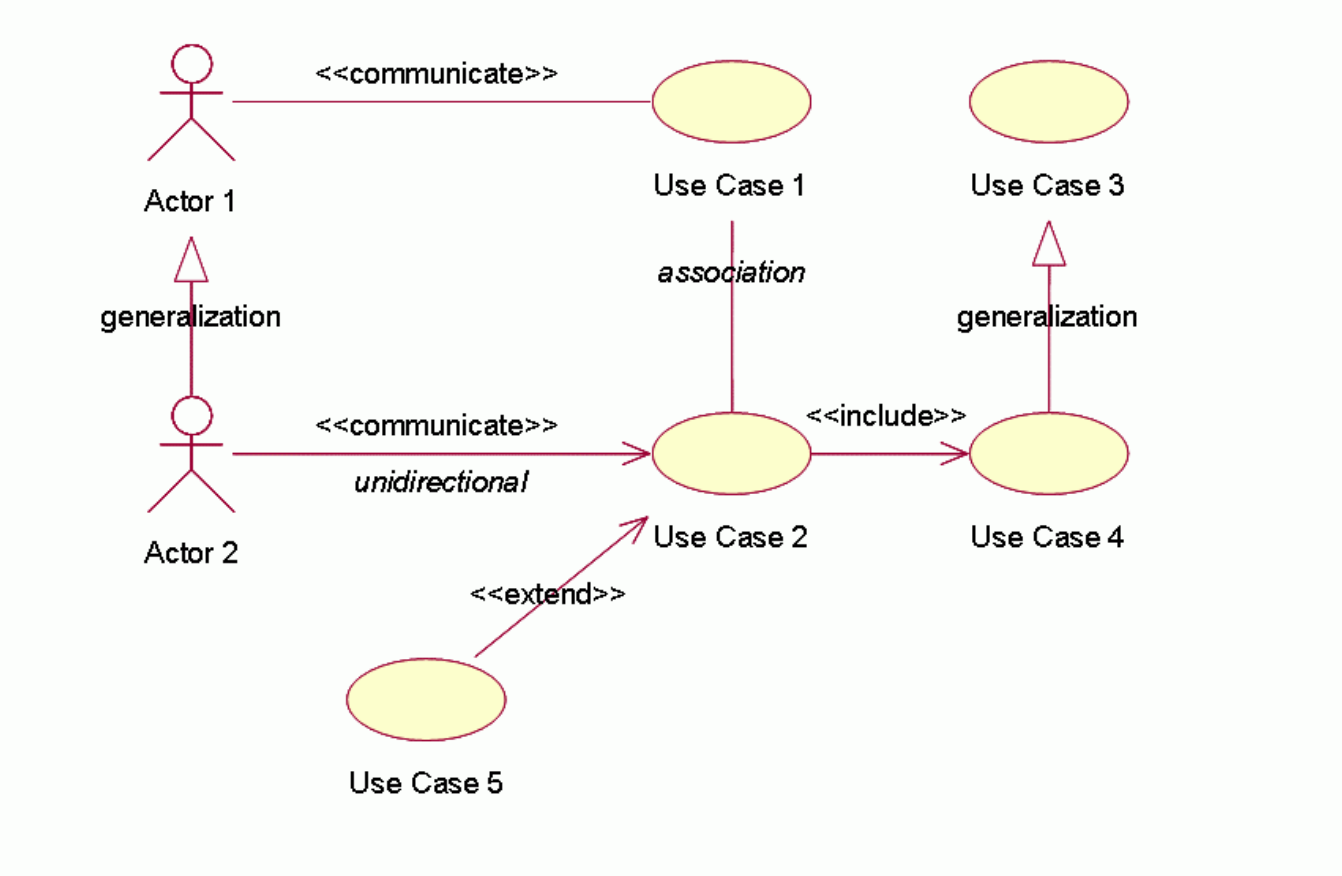
Proiectarea Diagramelor de Clase



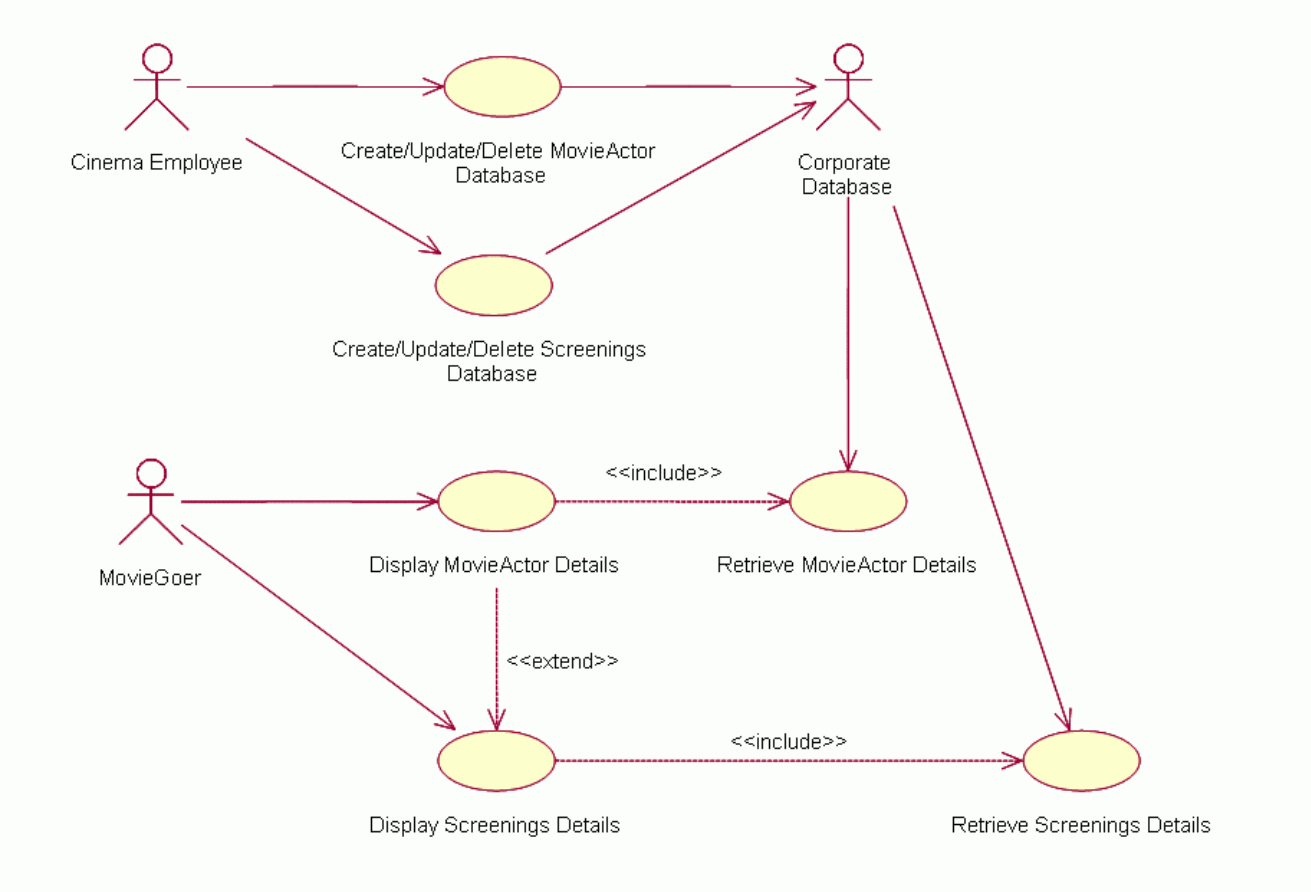
Diagrame Use Case

- Principala tehnică UML de modelare la nivel de analiză a comportamentului
- Expresivitatea acestor diagrame nu rezidă neapărat în reprezentarea grafică. Puterea reală a acestor diagrame stă mai ales în specificațiile textuale asociate
- Diagramele ***Use Case*** (cazuri de utilizare) reprezintă o piesă importantă în definirea funcționalității sistemului
- Un Actor este un rol pe care cineva sau ceva îl joacă relativ la un Use Case.
  - Actorul comunică cu un caz de utilizare (prin relația de «***communicate***») şi aşteptă o reacție de la acesta – o valoare sau un alt rezultat observabil

Elemente de Modelare Use Case



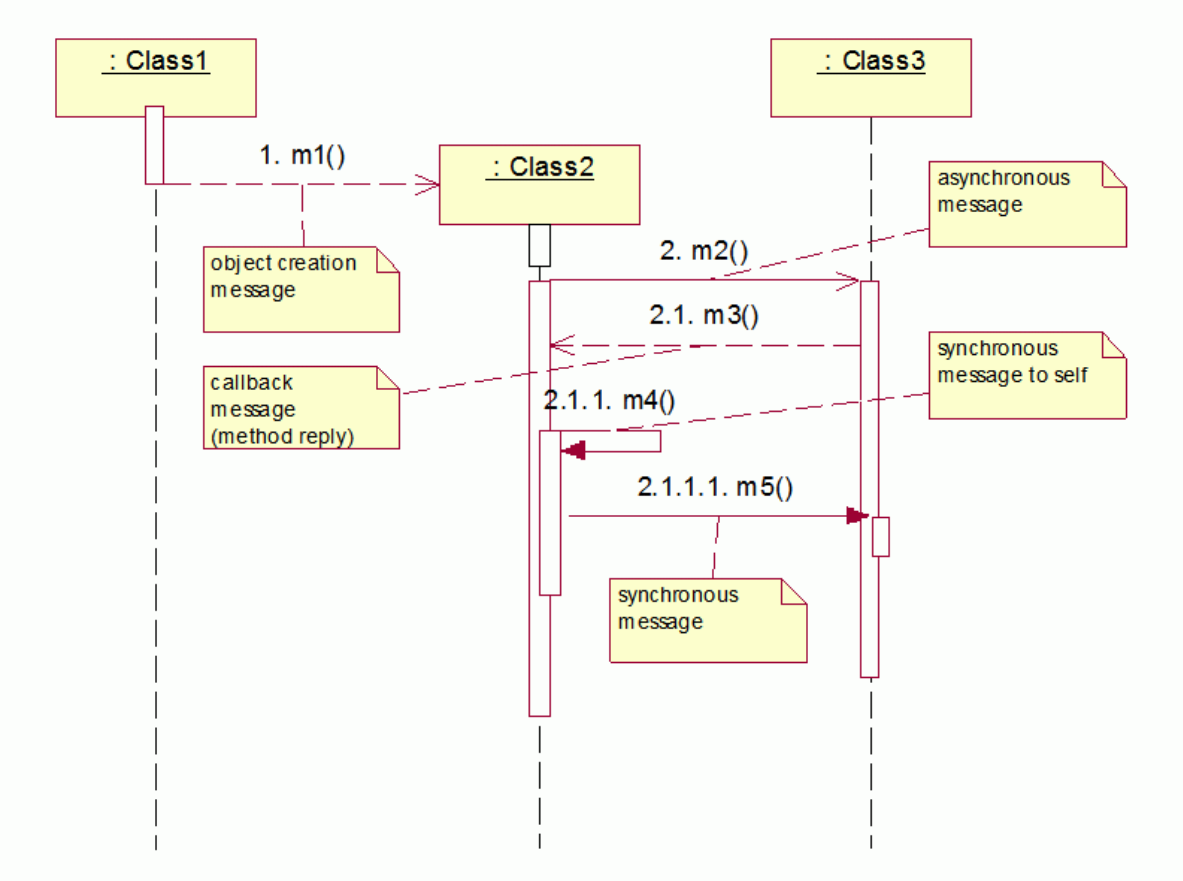
Diagrame Use Case. Exemplu



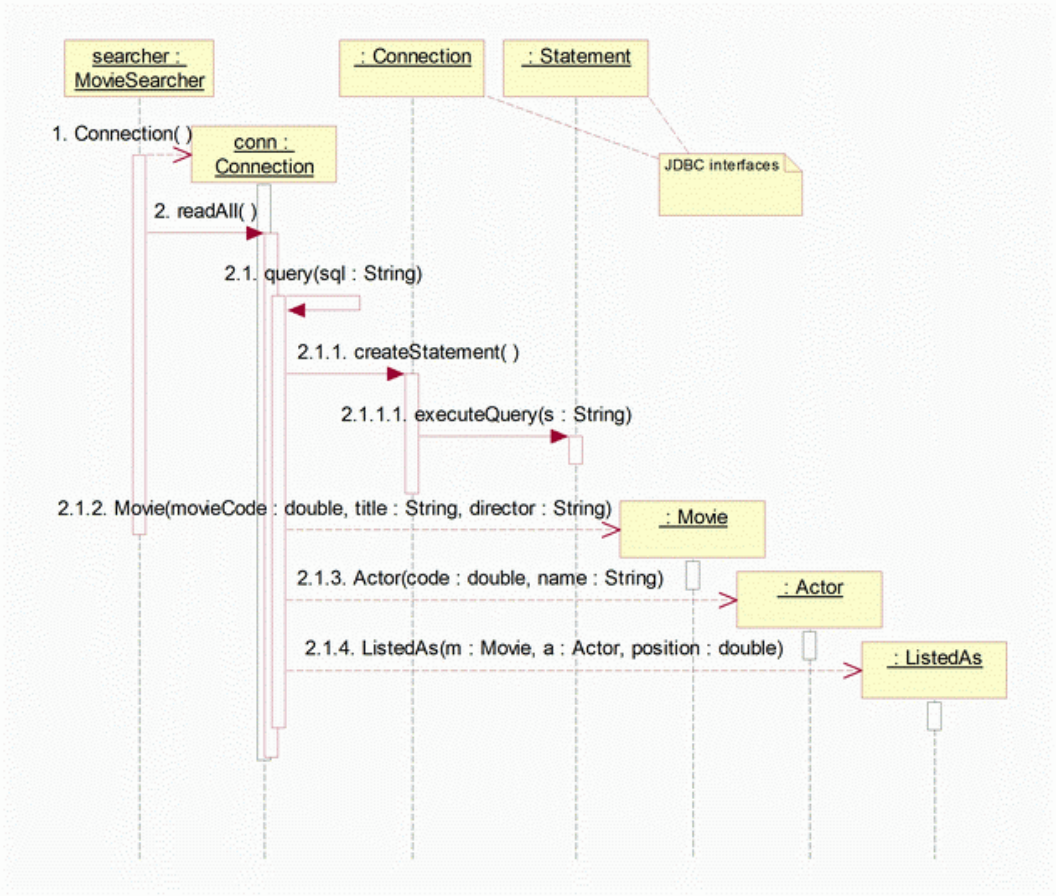
Diagrame Sequence

- Diagramele **Interaction** reprezintă principala tehnică de modelare la nivel de proiectare a comportamentului
- Există două tipuri de diagrame **Interaction**: **Sequence** şi **Collaboration**
- Diagramele **Sequence** (diagramele de secvență) sunt reprezentări grafice a secvenței mesajelor dintre obiecte
  - Secvența poate fi reprezentată prin plasarea mesajelor unul sub celălalt
  - Opțional se poate recurge şi la o numerotare a mesajelor care să denote ordinea lor
- Obiectul care recepționează un mesaj va activa metoda corespunzătoare acestuia
- Timpul în care fluxul de control ajunge la un obiect se numeşte **activare**
  - Se reprezintă printr-un dreptunghi ascuțit pe linia de viață a obiectului

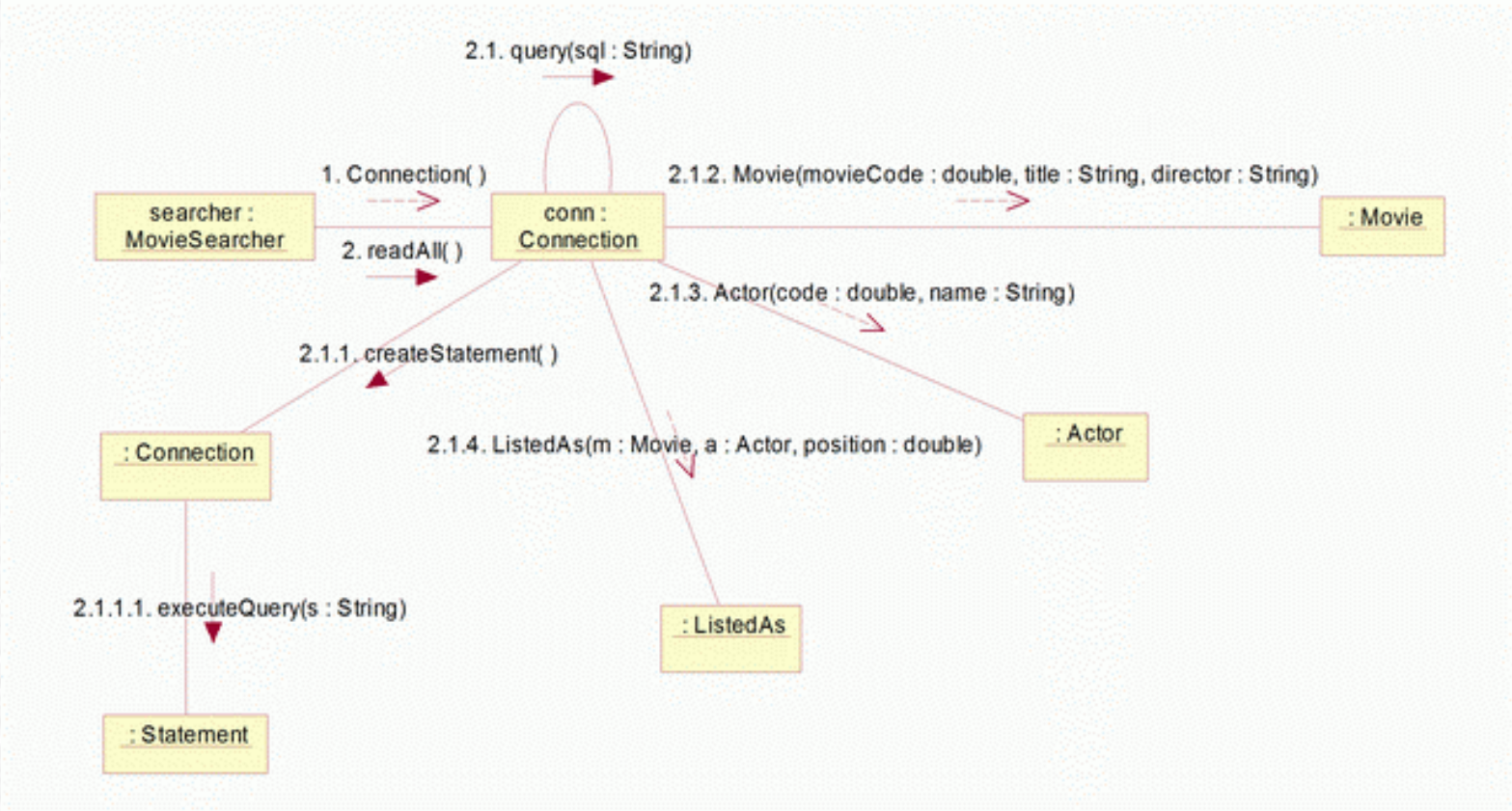
Diagrame Sequence. Reprezentarea Mesajelor



Diagrame Sequence. Exemplu



Diagrame Collaboration (Communication)

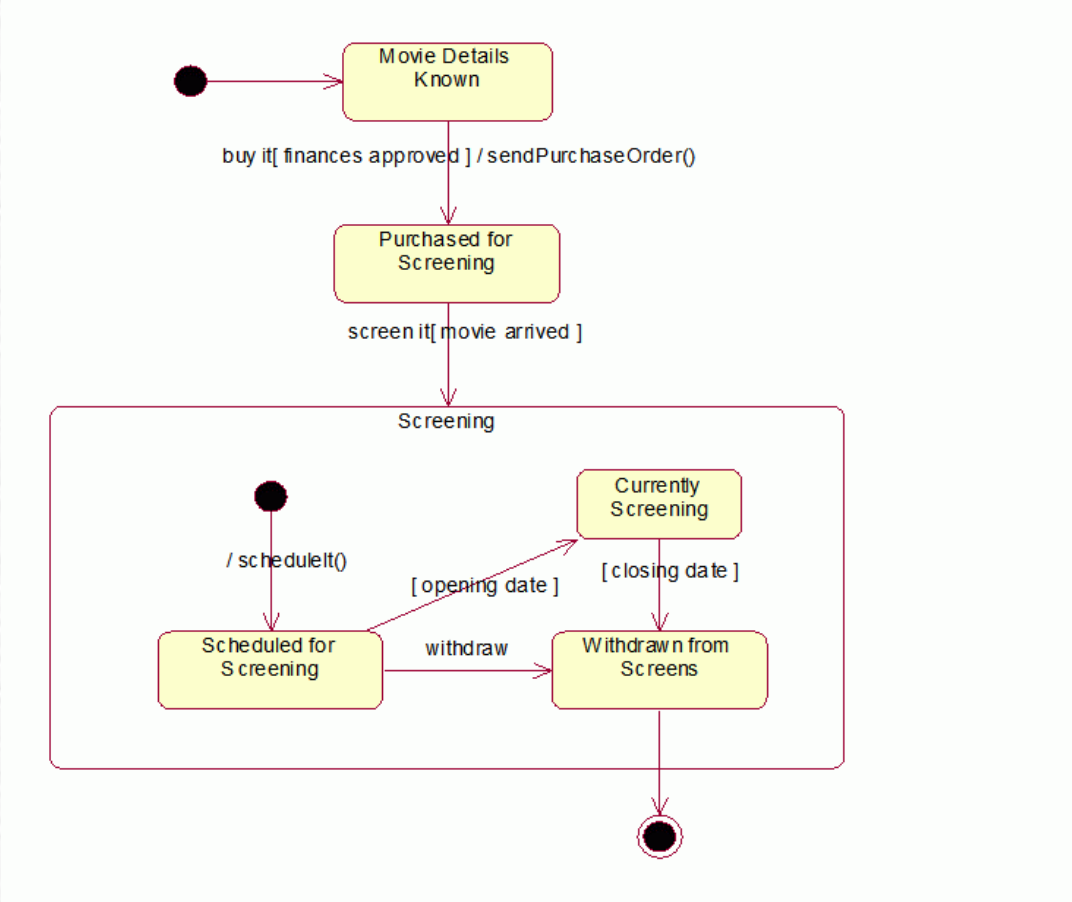


Diagrame Statechart

- Diagramele **Statechart** nu sunt specifice modelării orientate pe obiecte
- Surprind **starea** unui obiect şi **acțiunile** care conduc la schimbarea stării acelui obiect
- Reprezentate pentru fiecare clasă în parte care are o schimbare de stare demnă de luat în seamă pentru model
- Starea unui obiect (instanță de clasă) se schimbă când se modifică valoarea unor atribute ale sale
- Starea are o anumită durată care corespunde cu intervalul de timp dintre două tranziții

  `event-signature` *`[guard-condition]`* `/action-expression`

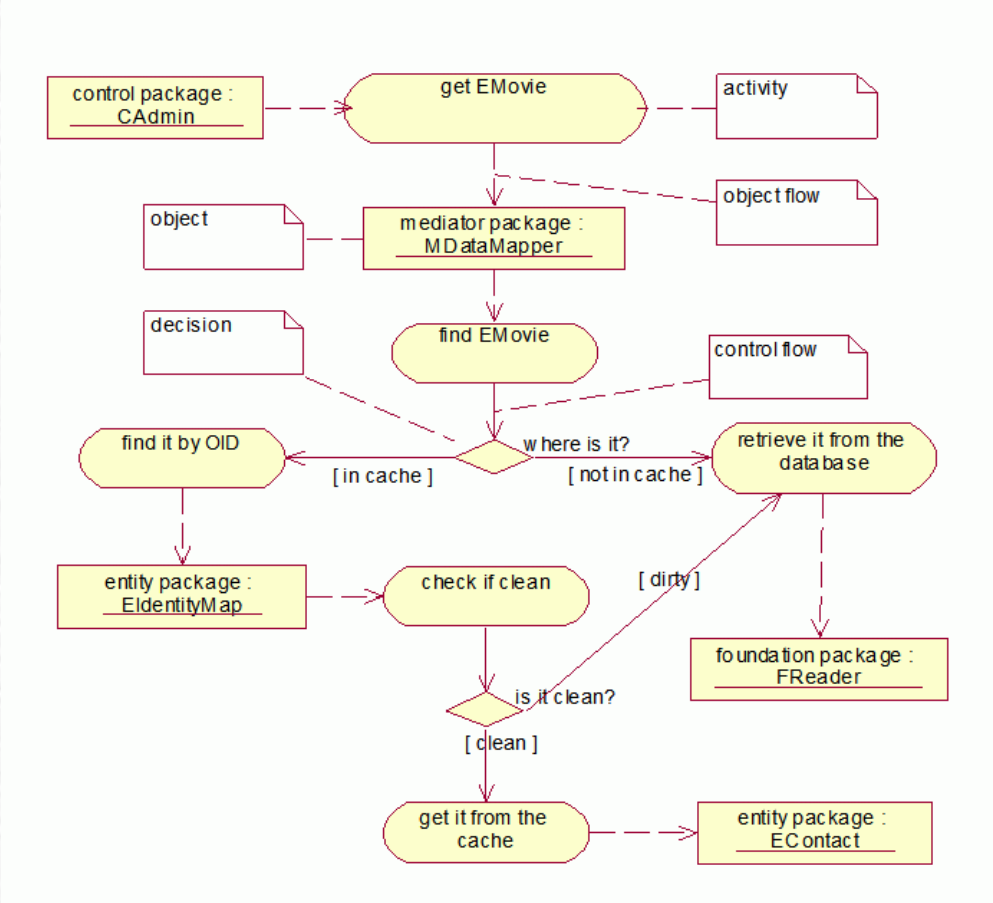
Diagrame Statechart. Exemplu



Diagrame Activity

- Diagramele **Activity** sunt automate cu stări care reprezintă un calcul, acțiunile executate şi tranzițiile rezultate din efectuarea acțiunilor
- În mod normal o diagramă **Activity** este ataşată unei implementări de operații sau unui caz de utilizare
- ***Action states*** (acțiunile de stare) sunt acțiuni care nu trebuie întrerupte de evenimente externe. Ele nu trebuie să aibă nici o tranziție de ieşire bazată pe evenimente explicite
- Tranzițiile de ieşire dintr-o acțiune de stare sunt rezultatul încheierii activităților din acea stare

Diagrame Activity. Exemplu



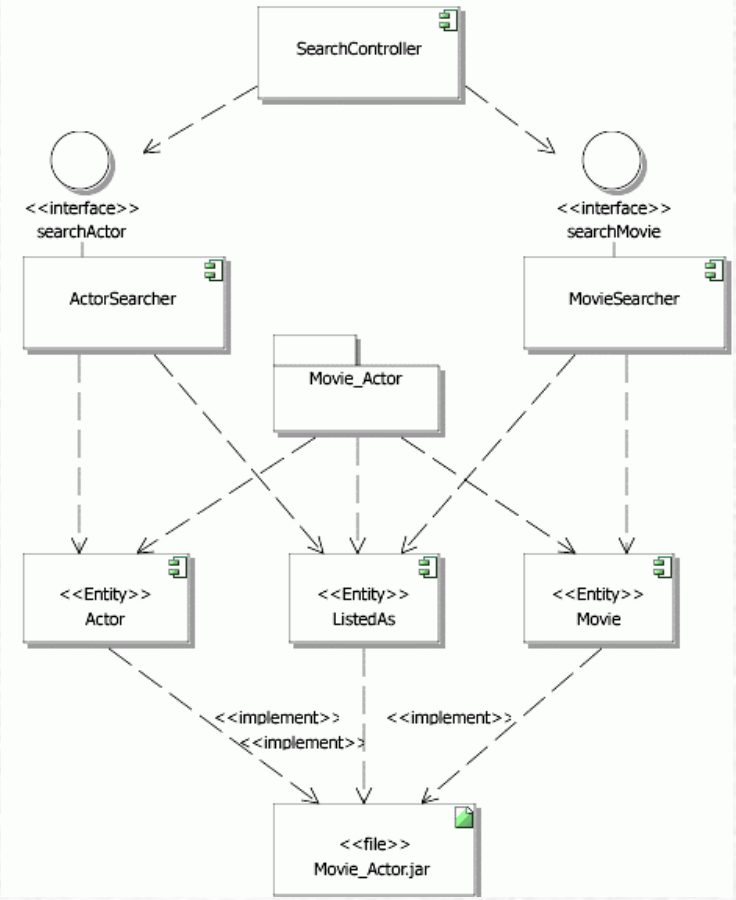
Diagrame Implementation

- Sunt modele pentru implementarea fizică a sistemului
- Evidențiază componentele sistemului, structura lor şi dependențele. De asemenea arată cum sunt ele plasate pe diverse calculatoare din rețea
- Există două tipuri de diagrame:
  - **Component**
  - **Deployment**
- Diagramele **Component** arată structura componentelor, inclusiv interfețele lor şi implementarea dependențelor
- Diagramele **Deployment** arată instalarea pentru rulare pe calculatoarele din rețea

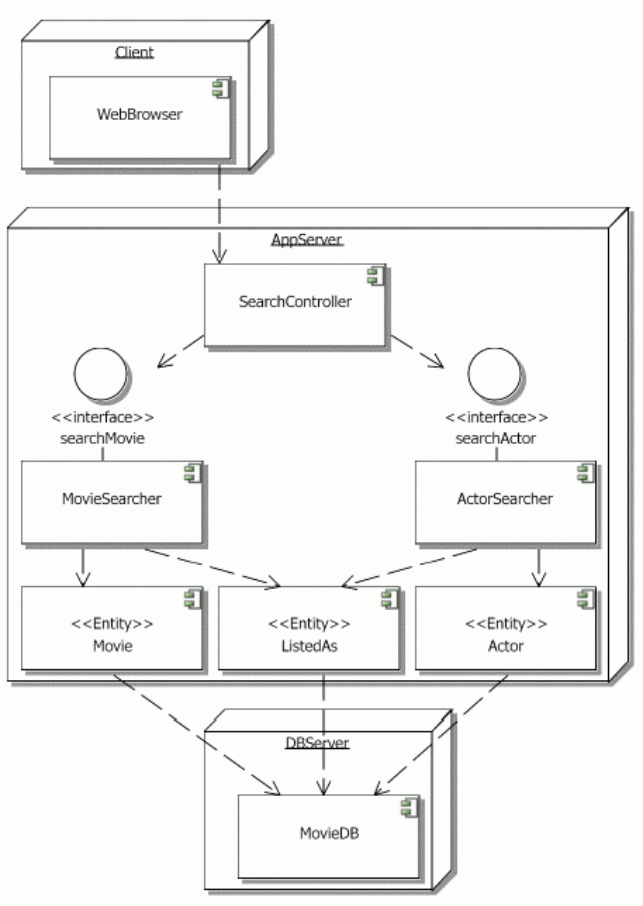
Diagrame Component

- *"A **component diagram** shows the dependencies among software components, including the classifiers that specify them (for example, implementation classes) and the artifacts that implement them; such as, source code files, binary code files, executable files, scripts." (UML, 2003b, p.3-169)*
- *"A **component** represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces." (UML, 2003b, p.3-174)*
- O componentă expune un set de interfețe care reprezintă serviciile oferite de elementele care compun componenta
- Componentele pot fi conectate la alte componente prin înglobarea lor fizică în acestea (relații de tip **«*reside*»** sau ***«implement»***)
- Pentru ilustrarea grupării componentelor se pot utiliza **Pachete**

Diagrame Component. Exemplu



Diagrame Deployment. Exemplu



Concluzii

- Limbajul **UML** este un limbaj de modelare standard pentru sistemele software orientate pe obiecte
- Limbajele potrivite pentru modelare structurată sunt: ***Data Flow Diagrams*** *(DFDs)*, ***Entity-Relationship Diagrams*** *(ERDs)* şi ***Structure Charts***
- **UML** include diferite tipuri de diagrame precum: **Class**, **Use Case**, **Interaction**, **Statechart**, **Activity** şi **Implementation**.
- Modelarea orientată pe obiecte în **UML** este centrată pe clase dar este condusă de diagramele **Use Case**

Concluzii

- Diagramele **Interaction** reprezintă principala tehnică de modelare la nivel de proiectare a comportamentului
- Diagramele **Statechart** surprind **starea** unui obiect şi **acțiunile** care conduc la schimbarea stării acelui obiect
- Diagramele **Activity** sunt automate cu stări şi reprezintă un calcul
- Diagramele **Implementation** sunt modele pentru implementarea fizică a sistemului