

Metode de dezvoltare software

Diagrame UML introducere +
diagrame UML de cazuri de utilizare

06.03.2017

Alin Ștefănescu

UML

UML... în practică



De ce “modelăm”?

- complexitatea e o problemă în dezvoltarea programelor.
- folosirea unor modele poate înlesni abordarea complexității.
- un **model** este o reprezentare abstractă, de obicei grafică, a unui aspect al unui sistem.
- acesta permite o mai bună înțelegere a sistemului și analiza unor proprietăți ale acestuia.

O scurtă istorie a modelării

- modele “statice” (structurale): apărute destul de devreme, ca desene, nu obiecte.
- modele “dinamice”: d.ex. flowcharts (Gilbreth 1921), automate finite (McCulloch-Pitts 1943), statecharts (Harel 1980), diagrame de secvențe (1990) etc.
- astfel, diverse tipuri de modele pentru sisteme (software) au fost studiate de-a lungul timpului.
- însă modelarea a devenit foarte vizibilă după introducerea paradigmei “orientate pe obiecte” (OO).

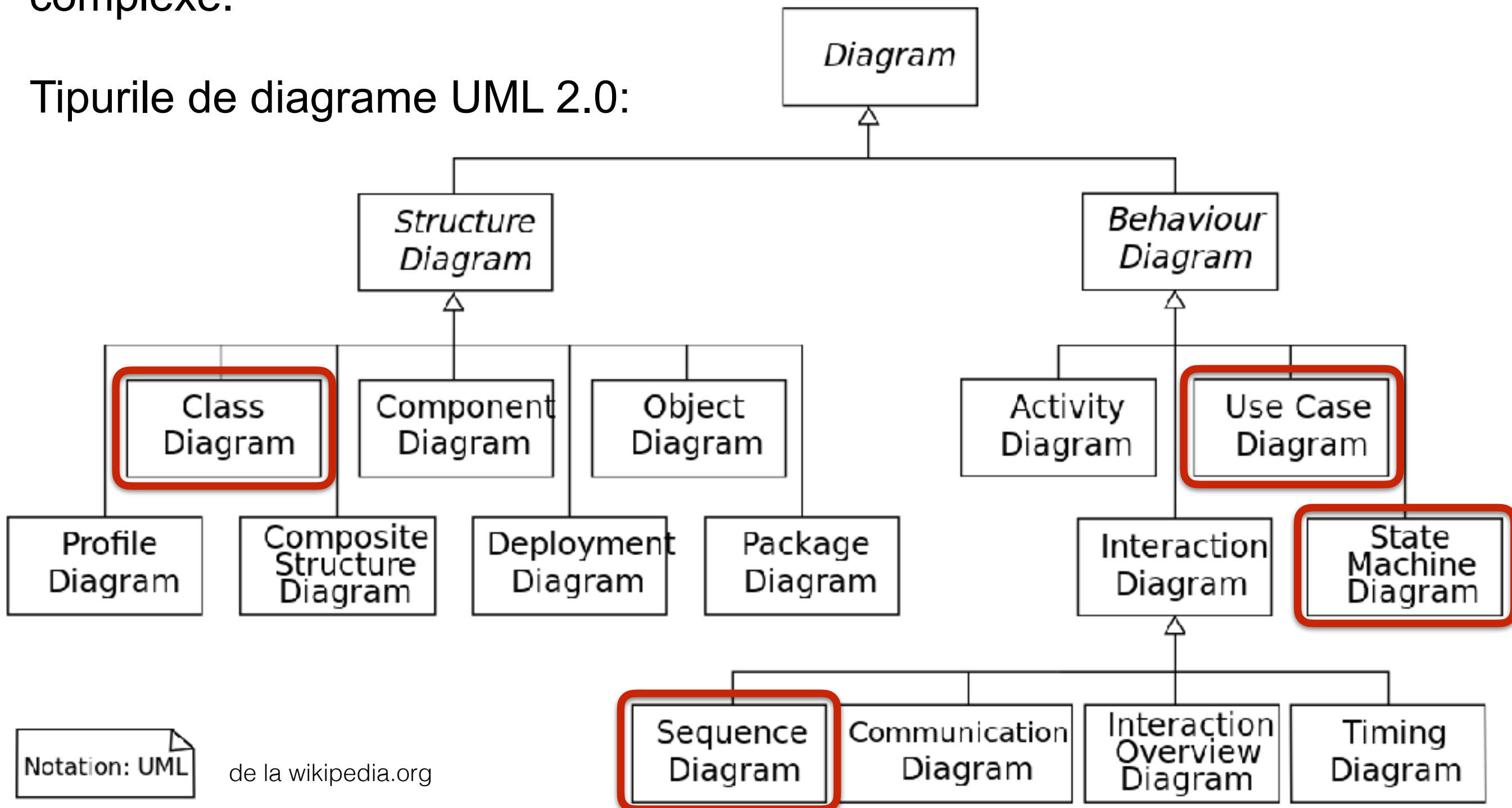
Modelarea orientată pe obiecte

- În anii '80 și începutul anilor '90 a avut loc o dezvoltare foarte puternică a paradigmei OO.
- o mulțime de experți OO, fiecare cu compania, tool-ul, cartea și modul de modelare propriu
- printre ei și Booch, Rumbauch, Jacobson, denumiți “the three amigos”, cei care au inițiat Unified Modeling Language (UML)
- Consorțiul OMG (Object Management Group) a reușit să standardizeze UML:
 - 1997 - UML 1.0
 - 2015 - UML 2.5

Cele 14 tipuri de diagrame UML

UML este un **limbaj grafic** pentru vizualizarea, specificarea, construcția și documentația necesare pentru dezvoltarea de sisteme software (OO) complexe.

Tipurile de diagrame UML 2.0:



Motive pentru care UML nu este folosit

- nu este cunoscută notația UML
- UML e prea complex (14 tipuri de diagrame)
- notațiile informale sunt suficiente
- documentarea arhitecturii nu e considerată importantă

Motive pentru care UML este folosit

- UML este standardizat
- existența multor tool-uri
- flexibilitate: modelarea se poate adapta la diverse domenii folosind “profiluri” și “stereotipuri”
- portabilitate: modelele pot fi exportate în format XMI (XML Metadata Interchange) și folosite de diverse tool-uri
- se poate folosi doar o submulțime de diagrame
- arhitectura software e importantă

Tipuri de folosire UML

UML e folosit în diverse moduri în proiecte sau organizații:

- diagrame UML pentru **a schița doar** diverse aspecte ale sistemului
- diagrame UML care **apar în documente** (uneori după ce a fost făcută implementarea)
- diagrame UML **foarte detaliate** sunt descrise în tool-uri înainte de implementare și apoi **cod este generat** pe baza acestor modele

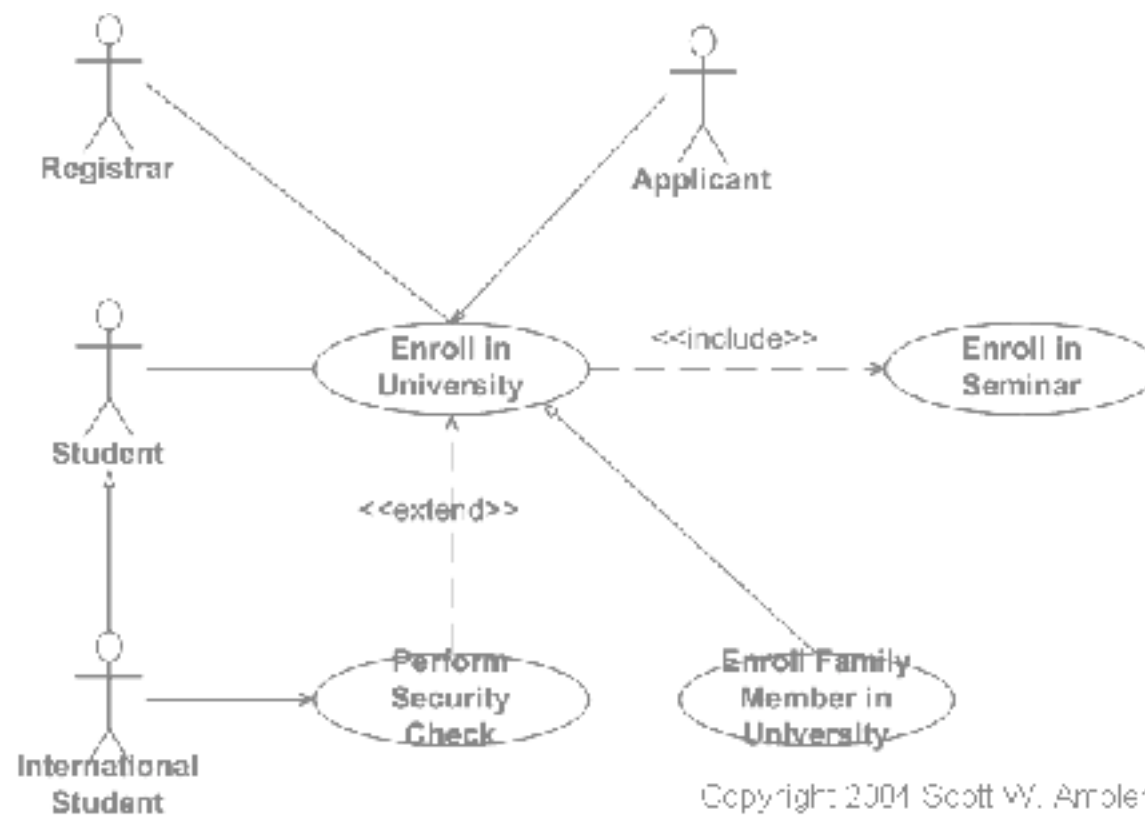
Tool-uri UML

Există foarte multe tool-uri pentru UML:

http://en.wikipedia.org/wiki/List_of_UML_tools

Câteva tool-uri gratuite (care mi s-au părut ok):

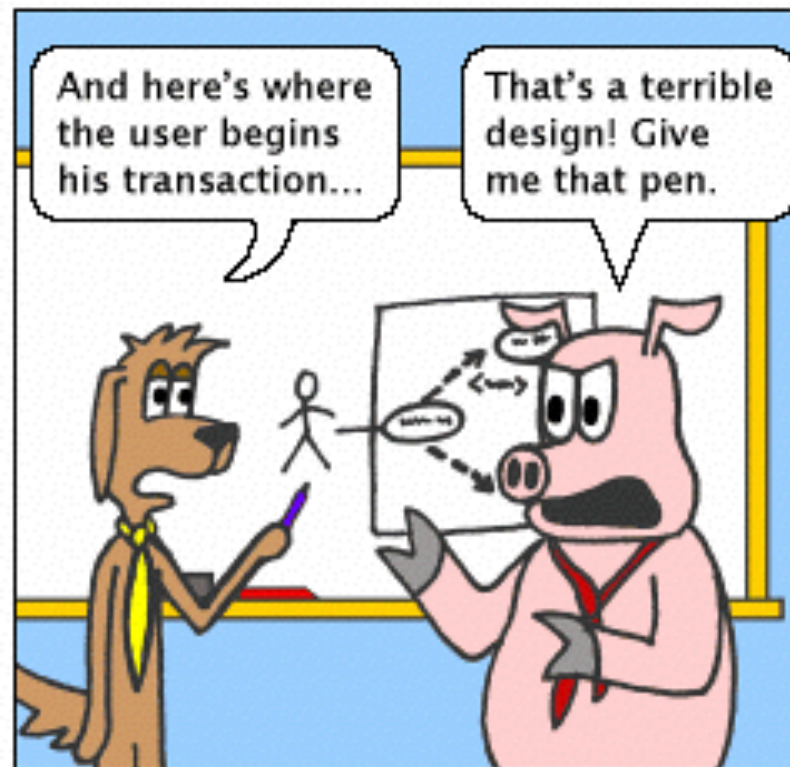
- **LucidChart, Creately, Gliffy** (web-based, mobile, etc.)
- **Visio 2016 Professional** (gratuit pentru studenți prin Dreamspark)
- **Astah** (community edition, gratuit)
<http://astah.net/download#community>
- **Visual Paradigm for UML** - (community edition, gratuit)
<http://www.visual-paradigm.com/download/community.jsp>



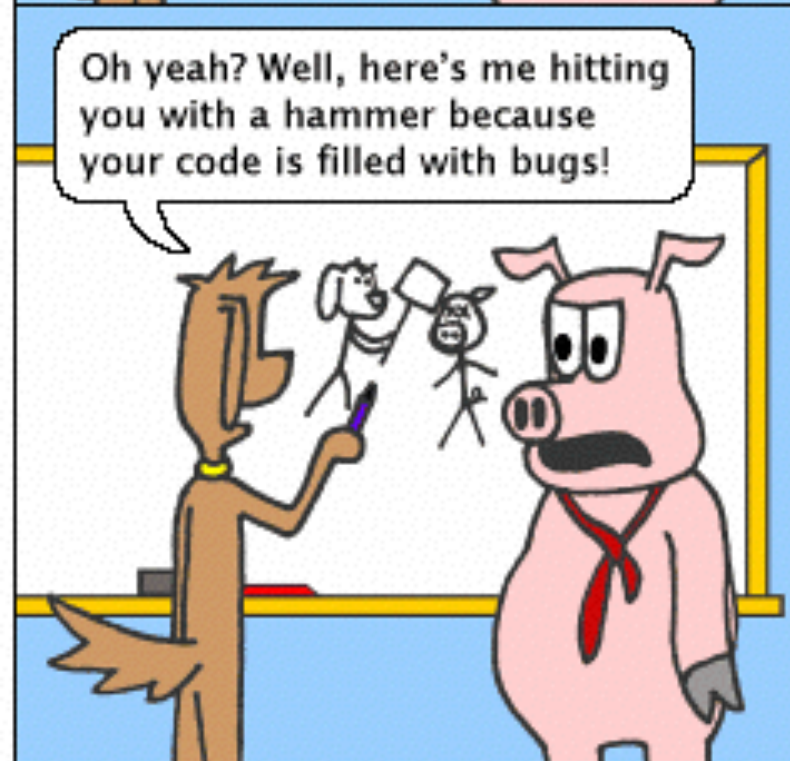
Cazuri de utilizare

Cazuri de utilizare... în practică

Hackles



By Drake Emko & Jen Brodzik



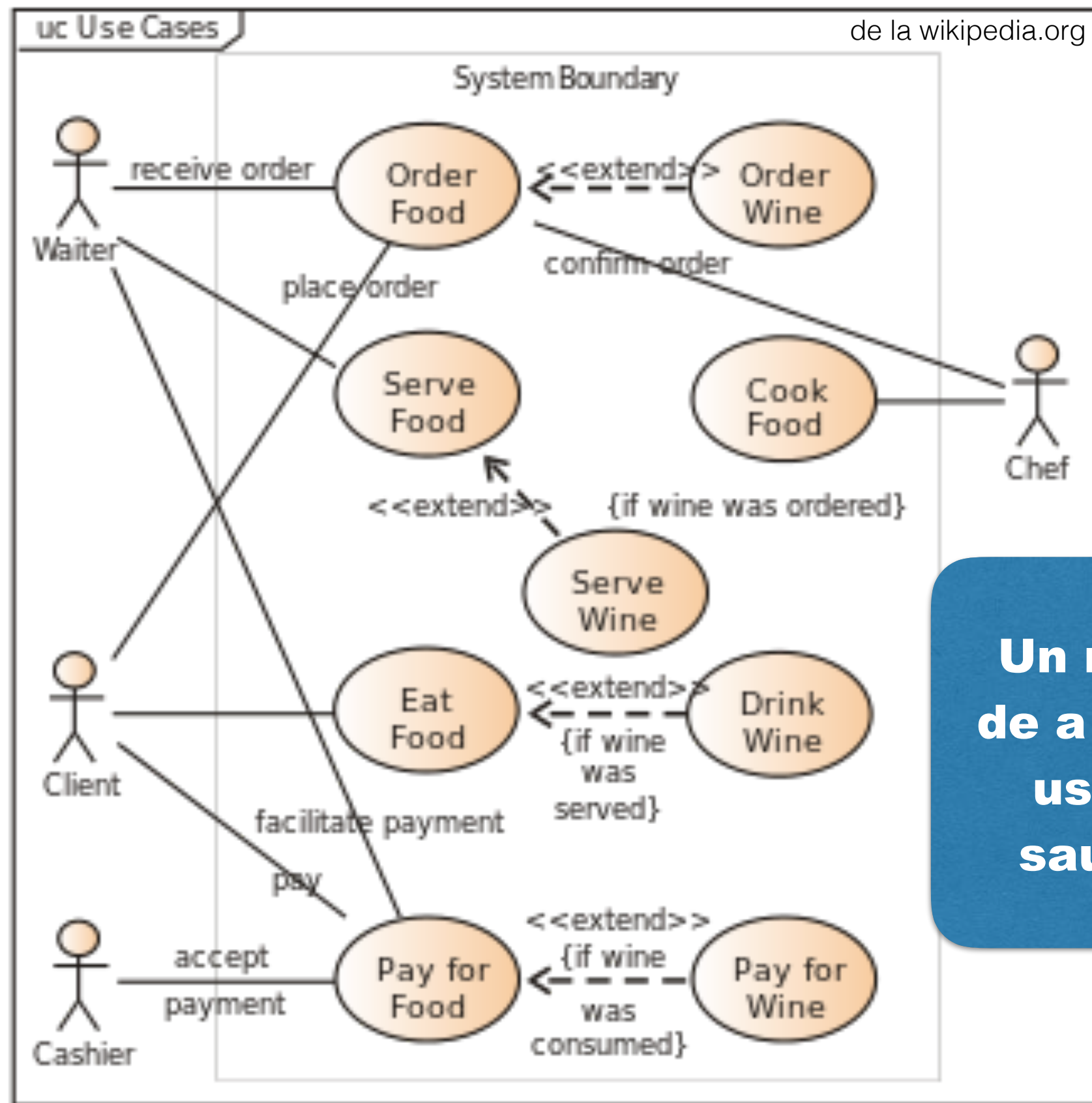
Cazuri de utilizare (Use Cases)

Introduse de Ivar Jacobson la începutul anilor '90.
Adoptate în standardul UML.

- Descriu comportamentul sistemului din punct de vedere al utilizatorului
- Două părți principale:
 - *sistem* (componente și descrierea acestora)
 - *utilizatori* (elemente externe)
- Cuprinde:
 - **Diagrama** cazurilor de utilizare
 - **Descrierea** cazurilor de utilizare

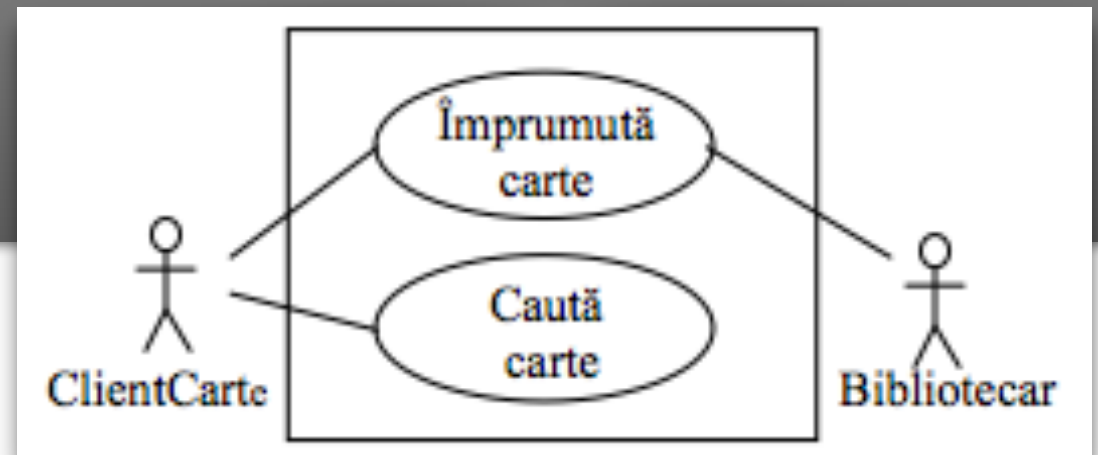


Diagrama cazurilor de utilizare - un exemplu



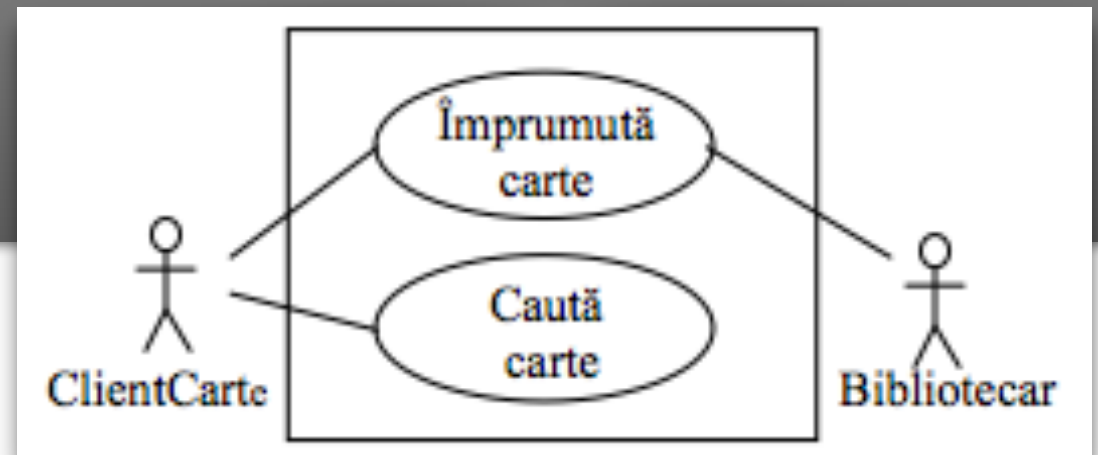
Un mod **vizual**
de a reprezenta
user stories
sau cerințele

Elementele principale



- **Caz de utilizare** (componentă a sistemului): unitate coerentă de funcționalitate sau task; reprezentată printr-un oval.
- **Actor** (utilizator al sistemului): element extern care interacționează cu sistemul; reprezentat printr-o figurină
- **Asociații de comunicare**: legături între actori și cazuri de utilizare; reprezentate prin linii solide
- **Descrierea cazurilor de utilizare**: un document (nativ) care descrie secvența evenimentelor pe care le execută un actor pentru a efectua un caz de utilizare

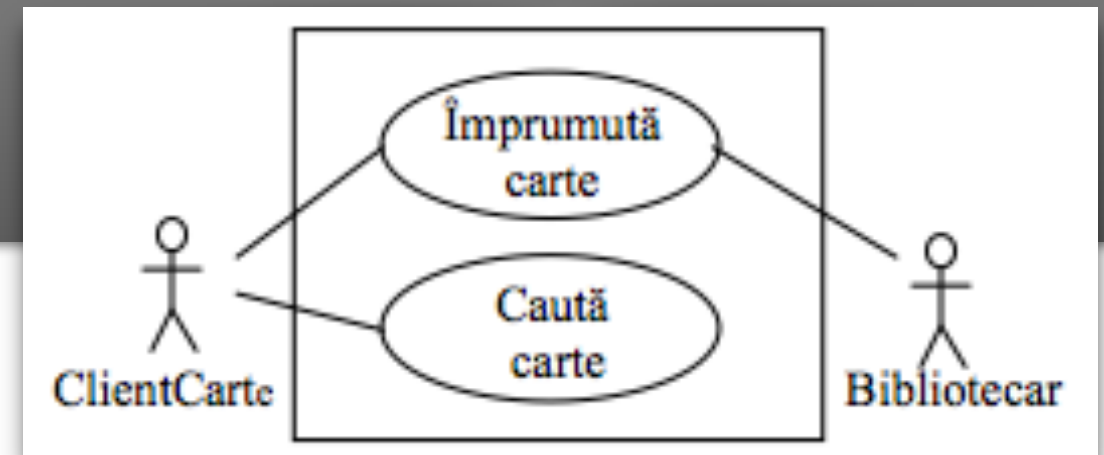
Actori



- Un actor reprezintă un **rol** jucat de un utilizator.
- Nu reprezintă un singur utilizator, ci o clasă de utilizatori.
- Același utilizator poate avea diferite roluri (d.ex. *Personal* sau *ClientCarte*), după cum un rol poate fi avut de mai mulți utilizatori.

Identificarea actorilor = identificarea rolurilor avute de utilizatori în cadrul sistemului

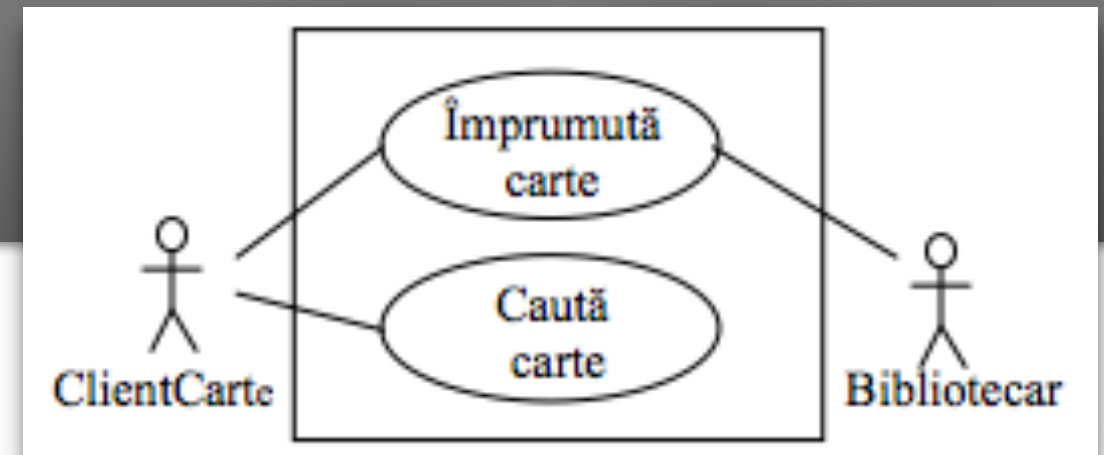
Clasificare actori



Pot exista actori primari și actori secundari:

- *Actorii primari* sunt cei pentru care folosirea sistemului are o anumită valoare (*beneficiari*); de exemplu *ClientCarte*. De obicei, actorii principali inițiază cazul de utilizare.
- *Actorii secundari* sunt cei cu ajutorul cărora se realizează cazul de utilizare; de exemplu un sistem pentru gestiunea unei biblioteci. Actorii secundari nu inițiază cazul de utilizare, dar participă la realizarea acestuia.
- Actorii pot fi atât actori umani cât și sisteme externe.

Cazuri de utilizare

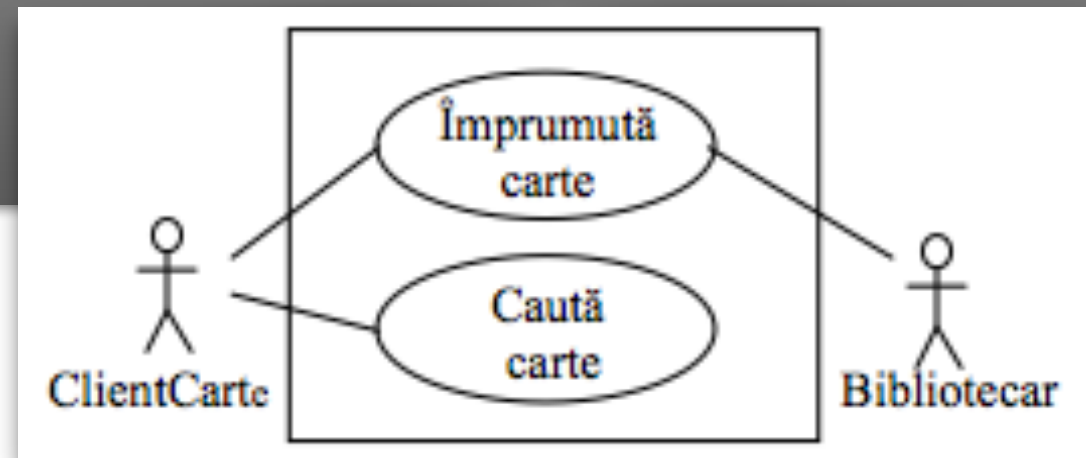


- Un caz de utilizare este o unitate coerentă de funcționalitate.
- Un caz de utilizare înglobează un set de cerințe ale sistemului care reies din specificațiile inițiale și sunt rafinate pe parcurs.
- Cazurile de utilizare pot avea complexități diferite; de exemplu “Împrumută carte” și “Caută carte”.

Cazuri de utilizare și scenarii

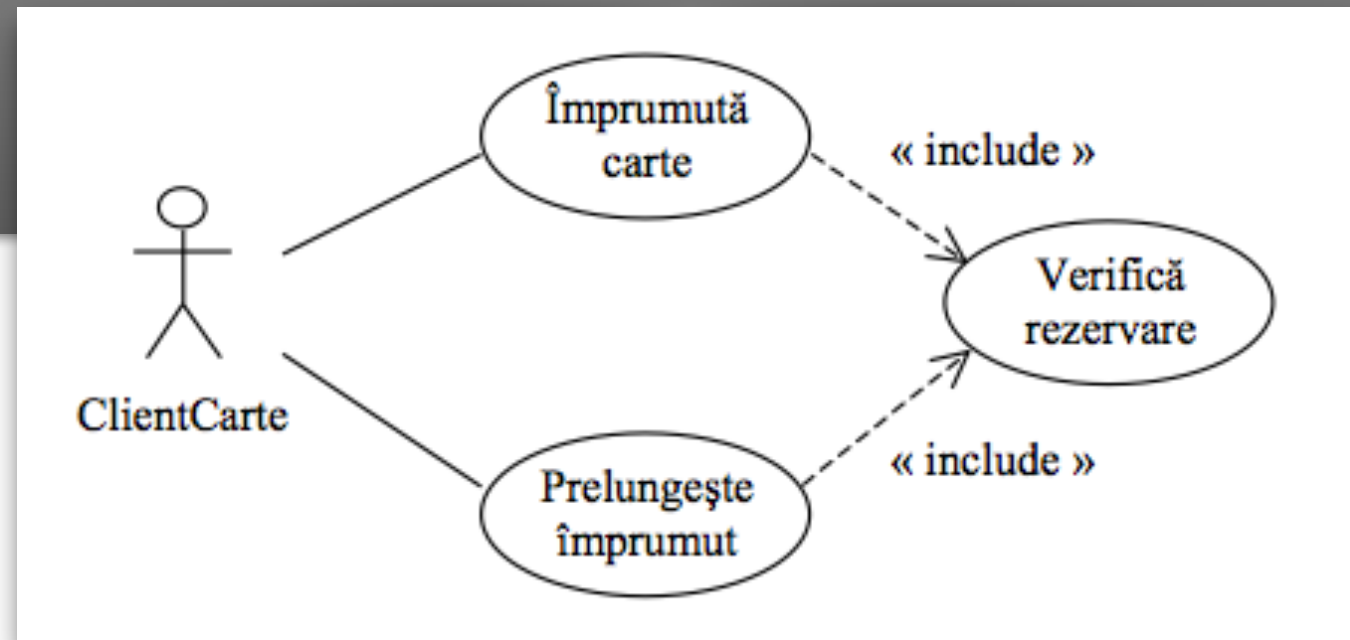
- un caz de utilizare reprezintă o mulțime de **scenarii** referitoare la utilizarea unui sistem.
- un scenariu este o instanță a unui caz de utilizare; de exemplu, următoarele două scenarii sunt instanțe ale cazului de utilizare “*Împrumută carte*”:
 - Persoana X împrumută o carte de la bibliotecă. Pentru că ea nu mai are cărți împrumutate, primește cartea și sistemul memorează acest lucru.
 - Persoana Y încearcă să împrumute o carte, dar este refuzată pentru că mai are deja împrumutate trei cărți, care este numărul maxim posibil.
- un caz de utilizare trebuie să fie însoțit de o descriere (un exemplu de șablon va fi furnizat mai târziu)

Frontiera sistemului



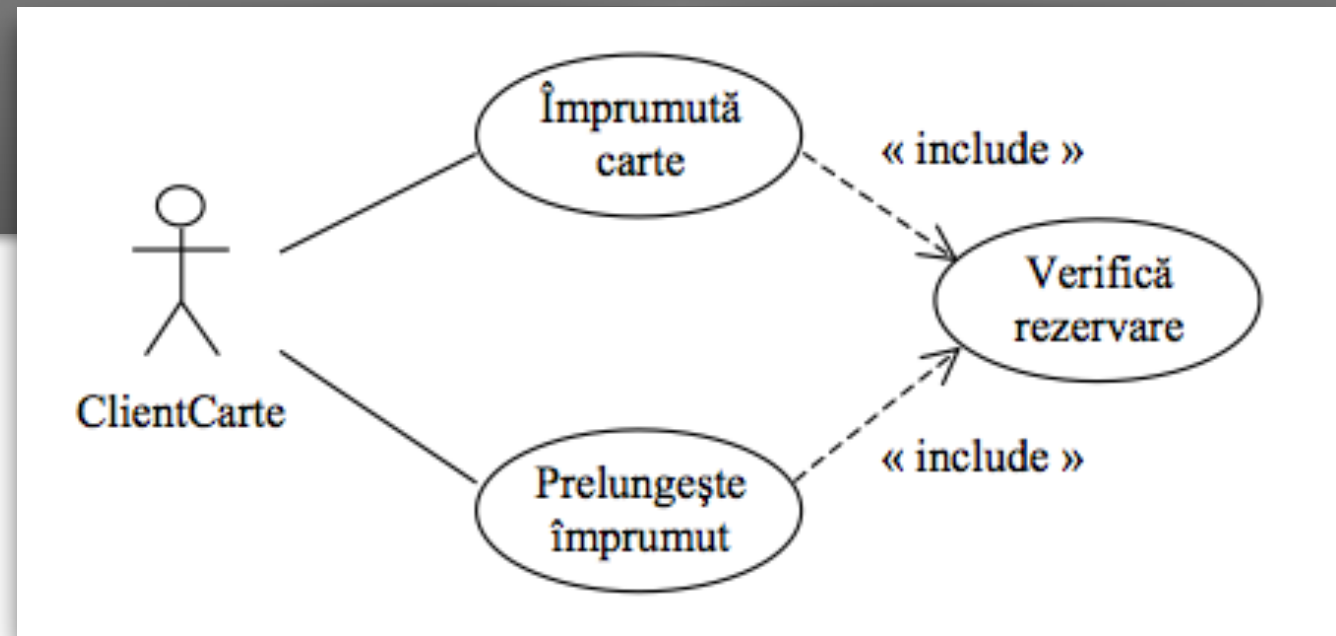
- este important de a defini frontiera sistemului astfel încât să se poată face distincție între mediul extern și mediul intern (responsabilitățile sistemului)
- ea poate avea un nume
- cazurile de utilizare sunt înăuntru, iar actorii în afară.
- dacă se dezvoltă un sistem software, frontiera se stabilește de obicei la frontiera dintre hardware și software.
- trasarea frontierei este opțională; trebuie însă indicată atunci când există mai multe subsisteme, pentru a le delimita clar.

Relația « include »



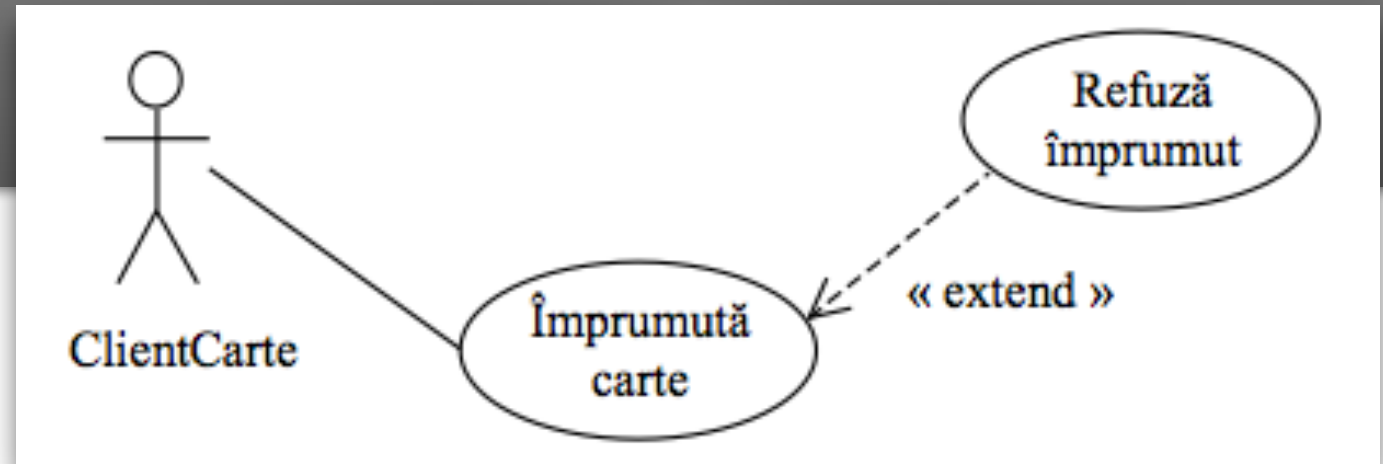
- Dacă două sau mai multe cazuri de utilizare au o componentă comună, aceasta poate fi reutilizată la definirea fiecăruia dintre ele.
- În acest caz, componenta refolosită este reprezentată tot printr-un caz de utilizare legat prin relația « include » de fiecare dintre cazurile de utilizare de bază.
- Practic, relația « include » arată că secvența de evenimente descrisă în cazul de utilizare inclus se găsește și în secvența de evenimente a cazului de utilizare de bază.

Relația « include »



- Săgeata este orientată către cazul de utilizare care este folosit și este etichetată cu « include ».
- Atunci când cazul de utilizare inclus se schimbă, acest lucru va afecta cazul de utilizare de bază, în schimb cazul de utilizare inclus nu va depinde de cazul de utilizare de bază.

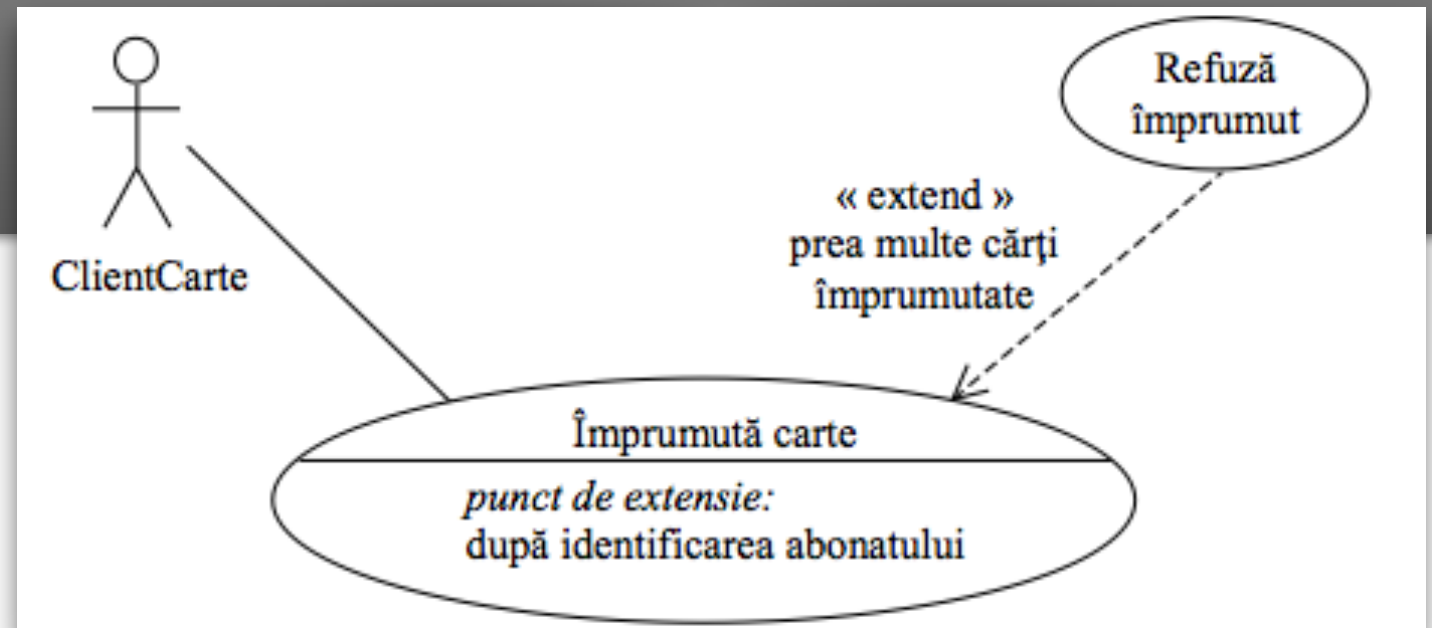
Relația « extend »



Relația « extend » se folosește pentru separarea diferitelor comportamente ale cazurilor de utilizare.

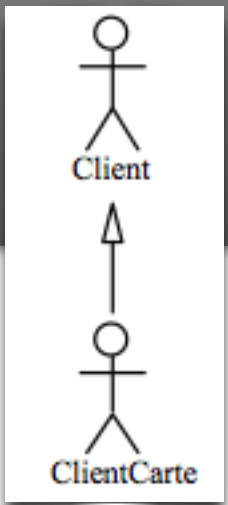
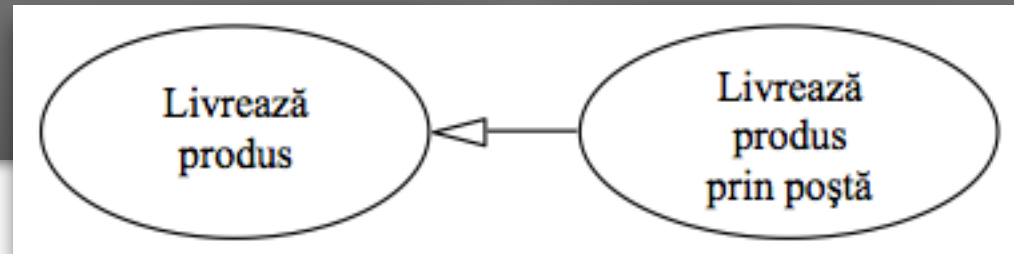
- Dacă un caz de utilizare conține două sau mai multe scenarii semnificativ diferite (în sensul că se pot întâmpla diferite lucruri în funcție de anumite circumstanțe), acestea se pot reprezenta ca un caz de utilizare principal și unul sau mai multe cazuri de utilizare excepționale.

Relația « extend »



- De obicei, se folosește pentru a pune în evidență **excepțiile**. D.ex. putem separa cazul de utilizare “Împrumută carte” într-un caz de utilizare frecvent întâlnit (principal), în care utilizatorului îi este permis să împrumute o carte și un caz de utilizare mai rar întâlnit (excepțional), în care utilizatorului îi este refuzat împrumutul deoarece depășește numărul maxim de cărți permise.
- Se poate specifica și “punctul de extensie”
- Atenție și la direcția săgeții (care arată spre cazul principal)

Relația de generalizare



Acest tip de relație poate exista atât între două cazuri de utilizare cât și între doi actori.

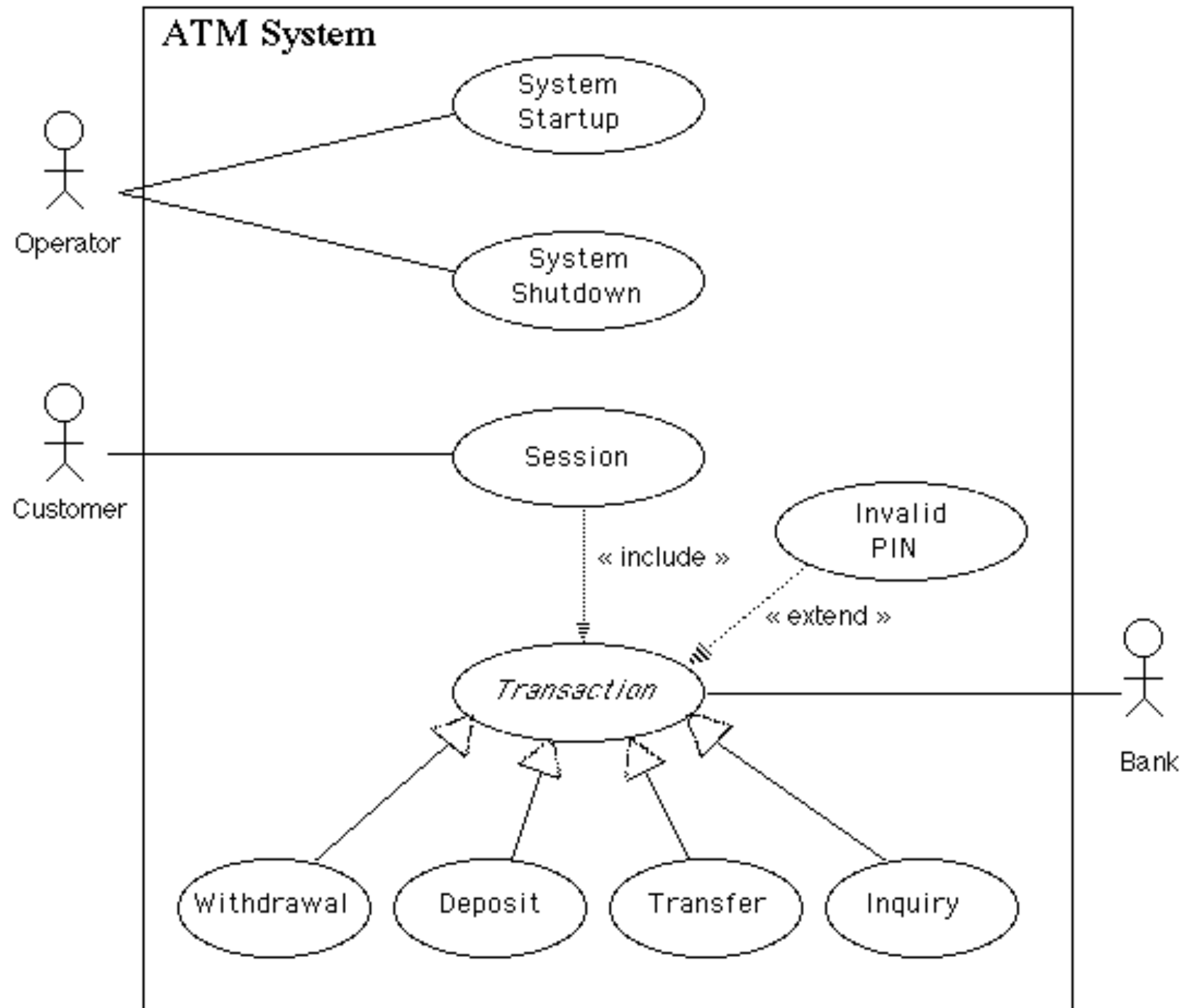
- generalizarea între cazuri de utilizare indică faptul că un caz de utilizare poate moșteni comportamentul definit în alt caz de utilizare.
- generalizarea între actori arată că un actor moștenește structura și comportamentul altui actor.

“Generalizarea” este asemănătoare cu relația « extend ». De obicei, folosim « extend » dacă descriem un comportament excepțional care depinde de o condiție testată în timpul execuției și “generalizarea” pentru a evidenția o anumită versiune a unui task.

Șablon pentru descrierea cazurilor de utilizare (o variantă)

Caz de utilizare	Identificatorul și numărul de referință al cazului de utilizare, istoria modificărilor
Descriere	Scopul în care e folosit cazul de utilizare și sursa cerințelor funcționale
Actori	Lista actorilor implicați
Ipoteze	Condiții ce trebuie să fie adevărate ca un caz de utilizare să se termine cu succes
Pași	Interacțiunile dintre actori și sistem care sunt necesare pentru a atinge scopul în care e folosit cazul de utilizare
Alternative (opțional)	Orice alternativă întâlnită în pașii cazului de utilizare
Cerințe non- funcționale (opțional)	Lista cerințelor non-funcționale pe care trebuie să le îndeplinească un caz de utilizare
Probleme	Lista problemelor care rămân să fie rezolvate

Încă un exemplu



Descriere la:

<http://www.math-cs.gordon.edu/courses/cs211/ATMExample/UseCases.html>

Atenție!

- **scopul principal** al acestui tip de diagrame este de a arăta **relația dintre actori și cazuri de utilizare**, nu relația dintre cazuri de utilizare (deși și aceasta este posibil prin relațiile *include* sau *extend* sau *generalizare*)
- nu se arată ordinea diferitelor cazuri de utilizare (aceasta se poate exprima prin alte diagrame - de stări sau de secvență).
- deci **NU există linii simple între cazuri de utilizare!**
(am observat această greșeală făcută de studenți destul de frecvent în decursul timpului)

Importanța cazurilor de utilizare

- Cazurile de utilizare descriu funcționalitatea sistemului; adică modul lui de folosire perceput de utilizatori (actorii externi).
- Scopul final al sistemului este de a realiza funcționalitatea descrisă în modelul cazurilor de utilizare (alături de cerințele nefuncționale).
- Cazurile de utilizare sunt folosite pentru:
 - analiză: identifică funcționalitatea cerută și o validează împreună cu clienții
 - design și implementare: trebuie realizate
 - testare: verifică sistemul; ele devin baza pentru generare de cazuri de testare.