Laborator 1

UML – Unified Modeling Language

Diagrama cazurilor de utilizare (*Use Case Diagram*)

Introducere

UML este un limbaj de modelare bazat pe notații grafice folosit pentru a *specifica, vizualiza, construi* şi *documenta* componentele unui program. UML este un limbaj cu ajutorul căruia se pot construi (descrie) modele. Un model surprinde un anumit aspect al unui program şi acelaşi model poate fi descris la diferite nivele de abstractizare. Fiecărui model îi corespunde o diagramă. Tipurile de diagrame existente în UML sunt:

- Diagrama cazurilor de utilizare (*Use Case Diagram*)
- Diagrama de clase (*Class Diagram*)
- Diagrame care descriu comportamentul:
  - Diagrame de interacțiuni (*Interactions Diagrams*)
    - Diagrama de secvență (*Sequence Diagram*)
    - Diagrama de colaborare (*Collaboration Diagram*)
  - Diagrama de stări (*State chart Diagram*)
  - Diagrama de activități (*Activity Diagram)*
- Diagrame de implementare:
  - Diagrama de componente (*Component Diagram*)
  - Diagrama de plasare (*Deployment Diagram*)

Fiecărei din cele trei mari faze din dezvoltarea un proiect software îi corespunde una sau mai multe diagrame UML şi anume:

- pentru faza de *analiza* se utilizează diagrama cazurilor de utilizare şi diagrama de activități;
- în faza de *analiză* se folosesc: diagrama de clase pentru precizarea structurii sistemului şi diagramele de stări şi interacțiune pentru descrierea comportamentului acestuia;
- în faza de implementare se utilizează diagramele de implementare.

Diagrama cazurilor de utilizare (*Use Case Diagram*)

Nici un program nu este izolat, el interacționând cu oameni sau cu alte sisteme pentru îndeplinirea unui scop.

O ***diagramă use case*** este una din diagramele folosite în UML pentru a modela aspectele dinamice ale unui program alături de diagrama de activități, diagrama de stări, diagrama de secvență şi diagrama de colaborare.

Elementele componente ale unei diagrame use case sunt:

- *use case*-uri;

- *actori;*

- *relațiile* care se stabilesc între use case-uri, între actori şi între use case-uri şi actori.

Use case-uri

Un *use case* (caz de utilizare) reprezintă cerințe ale utilizatorilor. Este o descriere a unei mulțimi de secvențe de acțiuni (incluzând variante) pe care un program le execută atunci când interacționează cu entitățile din afara lui (*actori*) şi care conduc la obținerea unui rezultat observabil şi de folos actorului. Un use case descrie ce face un program sau subprogram, dar nu precizează nimic despre cum este realizată (implementată) o anumită funcționalitate.

Fiecare use case are un *nume* prin care se deosebeşte de celelalte use case-uri. Acesta poate fi un şir arbitrar de caractere, însă de regulă numele sunt scurte fraze verbale care denumesc un comportament ce există în vocabularul sistemului ce trebuie modelat.

Figura 1 prezintă notația grafică pentru use case.

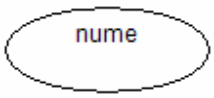


Figura 1: Notația grafică pentru use case

Comportamentul unui use case poate fi specificat descriind un flux de evenimente într-un text suficient de clar ca să poată fi înțeles de cineva din exterior (de exemplu utilizatorul). Acest flux de evenimente trebuie să includă cum începe şi se termină use case-ul atunci când acesta interacționează cu actori, ce obiecte sunt interschimbate, precum şi fluxuri alternative ale acestui comportament. Aceste fluxuri de evenimente reprezintă scenarii posibile de utilizare a sistemului.

Identificarea use case-urilor se face pornind de la cerințele utilizatorului şi analizând descrierea problemei.

Actori

Un *actor* reprezintă idealizarea unei peroane, proces sau obiect exterior care interacționează cu un sistem, subsistem sau o clasă. Actorii sunt entități exterioare sistemului. Ei pot fi utilizatori

(persoane), echipamente hardware sau alte programe. Fiecare actor are un *nume* care indică rolul pe care acesta îl joacă în interacțiunea cu programul.

Notație grafică pentru un actor este ilustrată în figura 2.

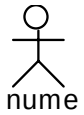


Figura 2: Notația grafică pentru actor

Există două moduri în care actorii pot interacționa cu un sistem:

- folosind sistemul, adică inițiază execuția unor use case-uri;
- sunt folosiți de către sistem, adică oferă funcționalitate pentru realizarea unor use case-uri.

Fiecare actor trebuie să comunice cu cel puțin un use case.

Pentru identificarea actorilor ar trebui să răspundem la următoarele întrebări:

- Cine foloseşte programul?
- De cine are nevoie programul pentru a-şi îndeplini sarcinile?
- Cine este responsabil cu administrarea sistemului?
- Cu ce echipamente externe trebuie să comunice programul?
- Cu ce sisteme software externe trebuie să comunice programul?
- Cine are nevoie de rezultatele (răspunsurile) oferite de program?

Relații

După cum am mai precizat, relațiile exprimă interacțiuni între use case-uri, între actori şi între use case-uri şi actori. Relațiile pot fi de mai multe tipuri: asociere, dependență şi generalizare.

Relația de asociere se defineşte între actori şi use case-uri, sau între use case-uri. Este folosită pentru a exprima interacțiunea (comunicarea) între elementele pe care le unește. Relația de asociere se reprezintă grafic printr-o linie şi este evidențiată în exemplele din figurile 3 şi 4.

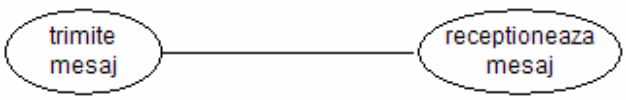


Fig. 3. Exemplu de asociere între use case-uri

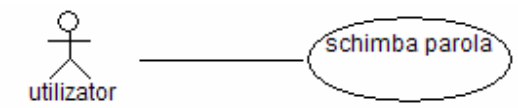


Fig. 4. Exemplu de asociere între actor şi use case

Relația de dependență se poate stabili numai între use case-uri. Acest tip de relație modelează două situații:

- cazul în care un use case foloseşte funcționalitatea oferită de un alt use case - dependența de tip *include*;
- există variante ale aceluiaşi use case – dependența de tip *extend*.

Dependența de tip ***include***. Notația grafică este dată în figura 5.

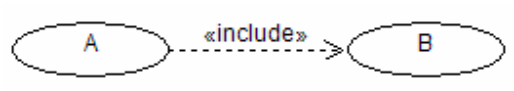


Fig. 5. Dependență de tip *include*

În acest caz comportamentul use case-ului B este inclus în use case-ul A. B este de sine stătător, însă este necesar pentru a asigura funcționalitatea use case-ului de bază A. În exemplul din figura 6, use case-ul `Stabileste grupul care lucreaza la campanie` are o relație de dependență de tip *include* cu use case-ul `Gaseste campanie`. Aceasta înseamnă că atunci când actorul `Manager campanie` utilizează `Stabileste grupul care lucreaza la campanie`, comportamentul use case-ului `Gaseste campanie` va fi inclus pentru a putea selecta o campanie relevantă.

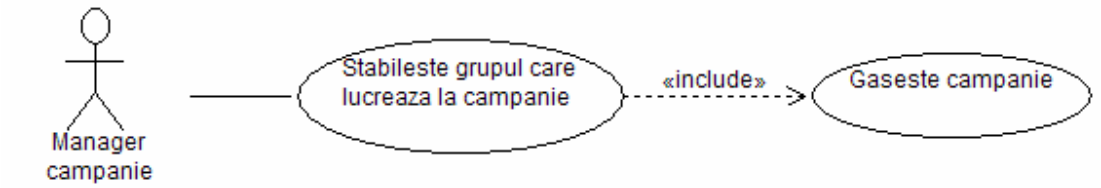


Fig. 6. Diagramă use case cu dependență de tip *include*

Dependența de tip *include* se foloseşte şi pentru a scoate în evidență un comportament comun (B poate fi inclus în mai multe use case-uri de bază – vezi figura 7).

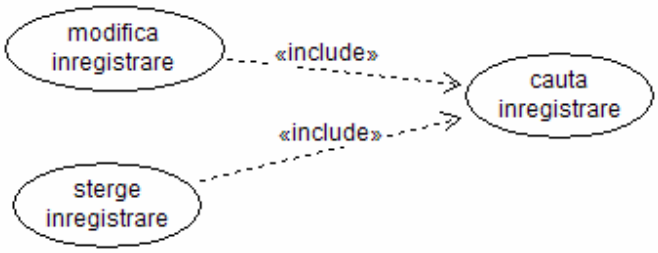


Fig. 7. Dependență de tip *include* în care un use case este inclus
în mai multe use case-uri de bază

Dependența de tip ***extend***. Notație grafică se poate vedea în figura 8.

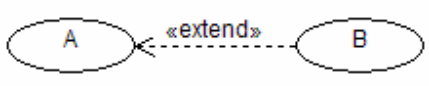


Fig. 8. Dependență de tip *extend*

În acest caz comportamentul use case-ului B poate fi înglobat în use case-ul A. A şi B sunt de sine stătătoare. A controlează dacă B va fi executat sau nu (vezi exemplul din figura 9).

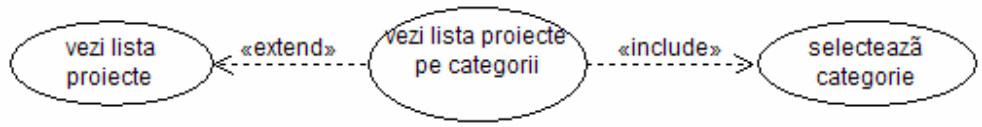
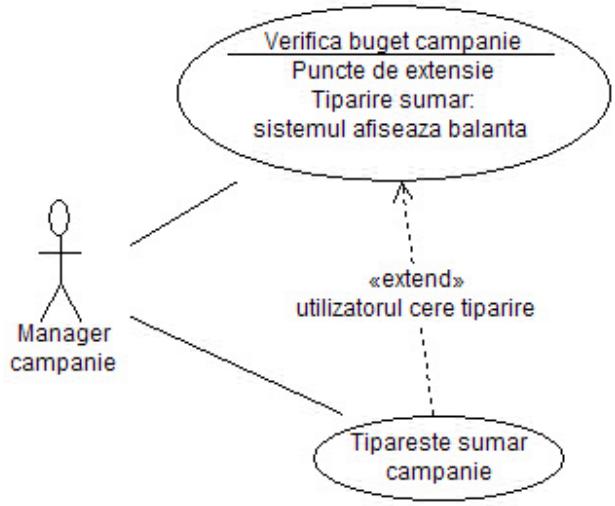
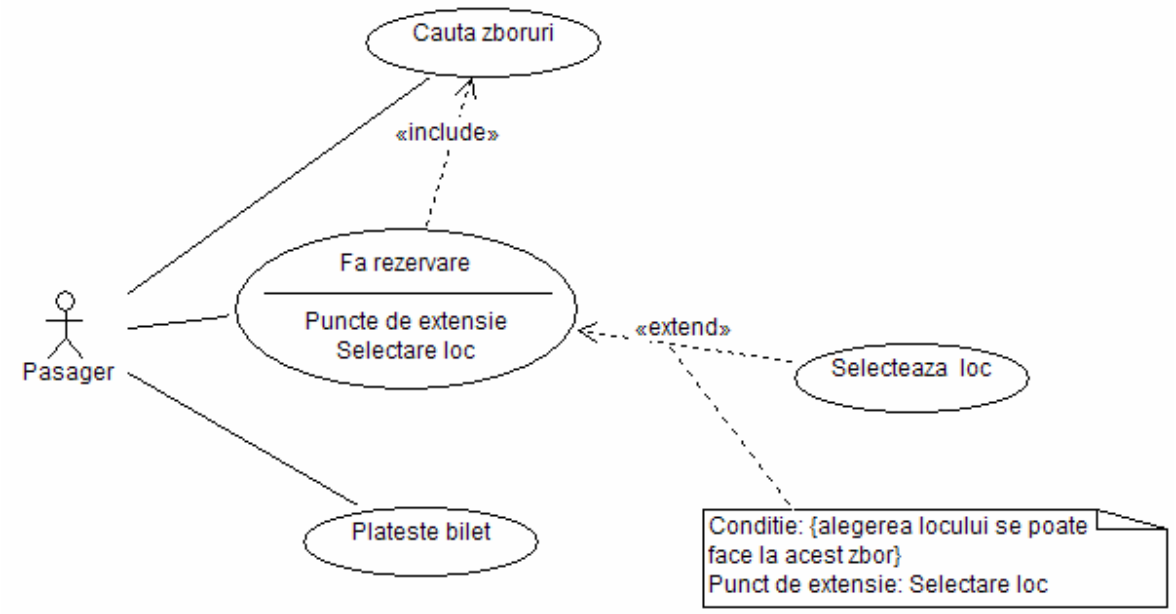


Fig.9. Exemplu de dependență de tip *extend*

Într-o dependență de tip *extend* pot apărea aşa numitele *punctele de extensie* care specifică locul în care use case-ul specializat (B) extinde use case-ul de bază (A). Pentru fiecare use case pot fi specificate mai multe puncte de extensie. Fiecare dintre aceste puncte trebuie să aibă un nume. Aceste nume trebuie să fie unice, însă nu este obligatoriu ca ele să coincidă cu numele use case-urilor specializate. De asemenea, trebuie precizată condiția de care depinde apelul use case-ului specializat. Acest tip de relație se foloseşte pentru a modela alternative. În figurile 10.a şi 10.b sunt prezentate diagrame use case cu dependențe *extend* care au puncte de extensie.



(a)



(b)

Fig. 10. Diagrame use case cu dependențe de tip *extend*

Relația de generalizare se stabileşte între elemente de acelaşi tip (doi actori, sau doua use case-uri). Este similară relației de generalizare (moştenire) care se stabileşte între clase. Figura 11 ilustrează notația grafică pentru relația de generalizare între use case-uri. Elementul derivat B moşteneşte comportamentul şi relațiile elementului de bază A. Use case-ul B este o specializare a use case-ului A.

În cazul unei relații de generalizare între use case-uri comportamentul poate fi modificat sau extins; use case-ul derivat B poate înlocui în anumite situații use case-ul de bază A. Case-ul derivat B controlează ce anume se execută şi ce se modifică din use case-ul de bază A.

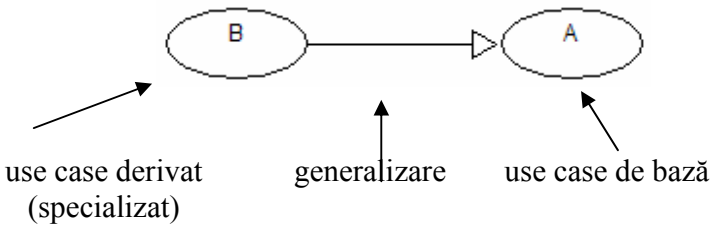


Fig. 11. Notația grafică pentru relația de generalizare între use case-uri

În figura 12 este prezentat un exemplu de relație de generalizare între use case-uri.

Fig. 12. Exemplu de relație de generalizare între use case-uri

După cum am precizat mai sus, relația de generalizare se poate aplica şi între actori. În exemplul din figura 13 este prezentată o relație de generalizare între actori.

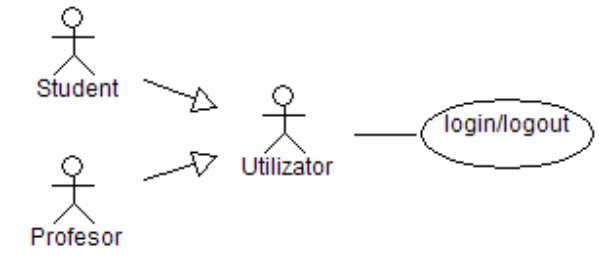
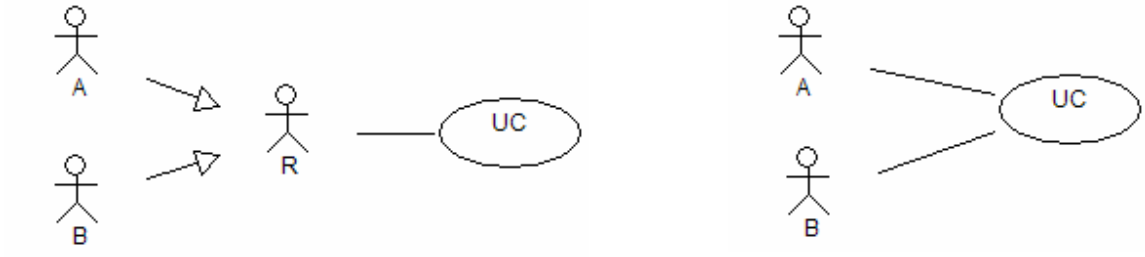


Fig. 13. Relație de generalizare între actori

În exemplul din figura 14.a actorii A şi B joacă acelaşi rol R atunci când comunică cu use case-ul UC şi joaca roluri diferite în interacțiunea cu use case-ul UC în figura 14.b.



(a) (b)

Fig. 14. Tipuri de roluri jucate de actori în interacțiunea cu use case-ul

În figura 15 este prezentată o diagramă use case care utilizează toate tipurile de relații definite anterior.

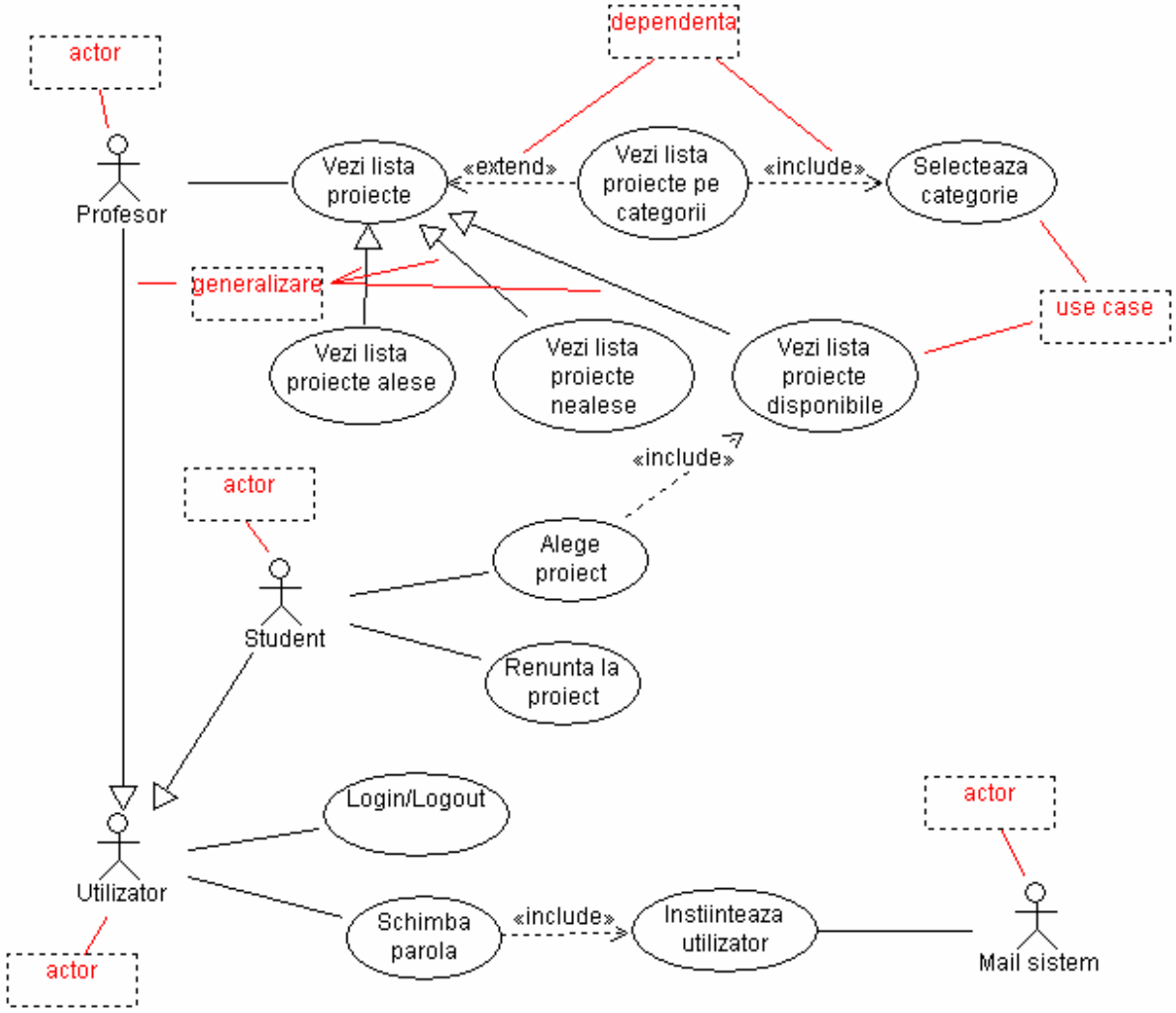


Fig. 15. Exemplu de diagrama use case

Reamintim utilizările diagramei use case:

- pentru a modela contextul unui sistem: determinarea graniţelor sistemului şi a actorilor cu care acesta interacţionează.
- pentru a modela cerinţele unui sistem: *ce* trebuie să facă sistemul (dintr-un punct de vedere exterior sistemului) independent de *cum* trebuie să facă. Va rezulta specificarea comportamentului dorit. Sistemul apare ca o cutie neagră. Ceea ce se vede este cum reacţionează el la acţiunile din exterior.

Probleme propuse

Pentru fiecare din problemele de mai jos să se identifice şi să se analizeze cerinţele utilizând diagrame use case:

1. Automat cafea (alegere tip cafea, introducere moneda, eliberare rest, preluare produs, etc)
2. ATM (verificare PIN, vizualizare suma din contul personal, extragere, tipărire chitanţa etc.)
3. Ceas electronic (afişare ora curentă / data curentă, modificare oră / dată, cronometru etc.)