PROIECTARE OBIECTUALĂ UML A UNUI SISTEM DE GESTIUNE A COSTURILOR UNEI ORGANIZAȚII PRIN METODA ABC – DIAGRAMELE CAZURILOR DE UTILIZARE ȘI ALE CLASELOR

Liana Anica-Popa

Academia de Studii Economice București

Rezumat: Etapa de proiectare a unui sistem de gestiune a costurilor prin metoda ABC (Activity Based Costing) este esențială pentru modul de funcționare al variantei finale.

Limbajul de modelare UML (Universal Modelling Language) este recunoscut ca fiind rezultatul efortului de unificare a celor mai bune practici din domeniul modelării sistemelor informatice.

Pentru a oferi o viziune cât mai completă asupra sistemului de gestiune vizat, este propus un set de diagrame UML. Fiecare diagramă oferă o perspectivă distinctă a sistemului, după cum sugerează și denumirile lor: diagrama cazurilor de utilizare; diagrama claselor, vizând trei aspecte: comportamentul acestora (diagrama de activități, diagrama de stări), interacțiunea între clase (diagrama de colaborări, diagrama de secvențe), implementarea acestora (diagrama componentelor, diagrama de amplasare).

Acest articol prezintă, pe scurt, caracteristici și instrumente utilizate în UML, urmate de diagramele cazurilor de utilizare și ale claselor. Într-un articol viitor, soluția de modelare, propusă pentru un sistem de gestiune ABC a costurilor la nivel de organizație va fi completată cu celelalte diagrame amintite.

Cuvinte cheie: UML, proiectare obiectuală, modelare orientată obiect, diagramă, caz de utilizare, diagrama cazurilor de utilizare, diagrama claselor, clasă, gestiune, costuri, metoda ABC.

1. Introducere

Limbajul UML (Universal Modelling Language) este cunoscut ca fiind rezultatul efortului de unificare a celor mai bune practici din domeniul modelării sistemelor informatice. UML constituie un mijloc de comunicare între echipa care implementează sistemul informațional și utilizatori sau alte persoane din interiorul unei organizații, oferind un cadru adecvat și pentru modelarea activităților de afaceri sau ale altor sisteme în termeni obiectuali. De asemenea, „UML furnizează și un set de mecanisme de extensie a conceptelor fundamentale, care să asigure personalizările sau adaptările necesare unei situații, probleme sau viziuni specifice ca și încorporarea, mai mult decât a probabilelor, evoluții în modelarea obiectuală"[1]. În plus, modelarea vizuală în UML este independentă de limbajul de programare sau de procesul de realizare, stabilite pentru sistemul care face obiectul studiului. UML „poate accepta și poate fi folosit în diverse metode, chiar dacă acestea aplică procese diferite"[2].

Ca orice limbaj, limbajul de modelare integrează un ansamblu de concepte, notații și un set de reguli de reprezentare și utilizare a acestora. Înseși conceptele de bază sunt modelate în UML, definire recursivă denumită metamodelare. Fiecare metamodel descrie elementele de modelare și reguli de compunere a acestora în alcătuirea modelelor. Calificarea limbajului UML drept „limbaj de vizualizare" și a modelării în UML ca fiind „vizuală" derivă din faptul că notația folosită include simboluri grafice ușor de interpretat.

Pentru a oferi o viziune cât mai completă asupra sistemului de obținut, UML propune realizarea unui set de diagrame care să faciliteze comunicarea atât în interiorul echipei de dezvoltare a produsului, cât și cu utilizatori ori alte persoane, neimplicate în mod direct în operațiile derulate de sistemul informațional vizat. Fiecare diagramă oferă o perspectivă distinctă a sistemului, după cum sugerează și denumirile lor:

- diagrama cazurilor de utilizare;
- diagrama claselor , vizând trei aspecte:
  1. comportamentul acestora;
     - diagrama de activități;
     - diagrama de stări:
  2. interacțiunea între clase;
     - diagrama de colaborări;
     - diagrama de secvențe:

---

[1] [17], p. 26;

[2] [17], p. 27;

3. implementarea acestora;
   - diagrama componentelor;
   - diagrama de amplasare.

Nivelurile unei arhitecturii UML sunt:

- meta-metamodel: acesta include conceptele de metaclasă, metaatribut, metaoperație;
- metamodel: instanță a unui meta-metamodel; în cadrul acestui nivel se lucrează cu noțiunile de clasă, atribut, operație;
- model: cu componente de tipul Cheltuieli, Inductor, CalculCostUnitar etc.;
- obiect-utilizator: incluzând instanțe ale unui model (de exemplu, clasa Inductor se instanțiază și se obține obiectul-utilizator <<100, „Număr comenzi", „ore">>).

Metamodelul este structurat în trei pachete: Fundamente pe care se bazează următoarele două: Comportament și Gestionar de modele. Pachetul Fundamente include componentele: Nucleu, Tipuri de date și Mecanisme de extensie.

Nucleul propune două categorii de componente:

- componente abstracte, neinstanțiabile: elemente de modelare, elemente generalizabile și clasificatori;
- componente instanțiabile, concrete: clase, atribute, operații și asocieri.

Diagramele amintite anterior folosesc în totalitate elemente de modelare, pentru care există reprezentări grafice, definiții semantice și reguli de utilizare bine precizate. Un element de modelare poate fi utilizat în mai multe tipuri de diagrame.

Între diferitele elemente de modelare pot exista mai multe tipuri de relații: asocieri, agregări, dependențe și generalizări, simbolizate grafic ca în figura 1.

Figura 1. Simboluri grafice pentru relații

Mecanismele de extensie din nucleul UML sunt: restricțiile, stereotipurile și valorile etichetate. Cel mai des utilizate – restricțiile – reprezintă expresii logice, referitoare la anumite elemente de modelare folosite. Asocierea unei valori unui nume conduce la definirea explicită a unor proprietăți și are ca rezultat o valoare etichetată. Stereotipurile regrupează, sub o denumire dată, mai multe elemente de modelare cu aceeași structură, care pot fi particularizate prin atașarea de valori etichetate și restricții specifice.

Pachetul Comportament include patru module: Comportament comun, în care sunt specificate conceptele fundamentale referitoare la dinamica elementelor de modelare utilizate în diagrame; în strânsă corelație cu funcționarea acestuia, există modulele Colaborări, Cazuri de utilizare și Mașini cu stări.

Ultimul pachet al unui metamodel, modulul de gestionare a modelelor, definește unitățile de structurare și regrupare a elementelor folosite în modelarea sistemelor informaționale: Pachetul, Modelul și Subsistemul.

Un pachet reprezintă o grupare de elemente identificată printr-o denumire. Utilizatorul este cel care stabilește atât criteriul de grupare, precum și tipul elementelor reunite. Elementele trebuie să se identifice printr-un nume unic în cadrul pachetului. Modelul – o subclasă a pachetului din metamodel – se definește ca o reprezentare a unor aspecte din realitate, dintr-o anumită perspectivă. Cum aceeași realitate poate fi privită din mai multe puncte de vedere, pentru fiecare ipostază se poate construi un model distinct. Un subsistem reprezintă o subclasă a pachetului și a clasificatorilor din metamodel, care definește o unitate comportamentală a sistemului fizic folosind o interfață și operații proprii. Subsistemul este compus din două părți: una de specificare și alta de realizare, interconectate printr-un set de colaborări. Un subsistem poate figura drept componentă a altor subsisteme.

Utilizarea limbajului UML în modelare conduce la o viziune complexă asupra tuturor fazelor realizării unui sistem informațional. Modelarea în UML trebuie să ilustreze modelele cerințelor, ale analizei, ale proiectării și ale implementării. Pentru aceasta, sunt utilizate diagrame de diferite tipuri. Fiecare model din fazele următoare aduce o perspectivă diferită asupra sistemului și adaugă noi elemente imaginilor precedente, obținându-se, în final, o viziune de ansamblu a stadiilor prin care va trece sistemul, de la demararea realizării sale până la exploatarea sa.

Notorietatea limbajului UML a determinat apariţia unor produse - program dedicate, de tip CASE, cum sunt Rose (al firmei Rational), Rocase (realizat la Universitatea Babeş-Bolyai, Cluj-Napoca), GDPro (de la compania Advanced Software Technologie), Visual Modeler (din binecunoscutul pachet Visual Studio al firmei Microsoft), care asigură:

- realizarea automată a diagramelor şi modelelor;
- exploatarea lor simultană de către mai mulţi utilizatori;
- realizarea unui tablou general al tuturor elementelor sistemului, în care figurează toate completările şi modificările ulterioare, pe baza căruia se poate genera automat documentaţie („repository")[3];
- generarea automată a codului de program sursă, pornind de la modelele create;
- generarea automată de diagrame, pornind de la analiza structurii codului sursă („reverse engeneering");
- verificarea concordanţei modelului cu codul sursă, prin combinarea ultimelor două funcţionalităţi („round-trip engeneering").

2. Diagrama cazurilor de utilizare

Pentru a delimita aria de cuprindere şi pentru a reprezenta setul de utilizări ale unui sistem informatic, se realizează diagrama cazurilor de utilizare. Pe lângă urmărirea concordanţei dintre cerinţele utilizatorilor şi implementarea sistemului informatic, diagrama cazurilor de utilizare va constitui, de asemenea, cadrul de bază pentru testarea şi întreţinerea sistemului realizat. Fiecare caz de utilizare, este iniţiat de către un utilizator extern al sistemului, denumit actor, şi descrie interacţiunile dintre acesta şi o anumită entitate aparţinând sistemului. O astfel de entitate poate fi un subsistem sau o clasă, ori chiar sistemul informatic însuşi. Un actor poate iniţia cazuri de utilizare pentru mai multe entităţi, în fiecare situaţie având roluri diferite. În acelaşi timp, unui caz de utilizare îi pot corespunde mai mulţi actori, fiecare cu un rol propriu. O entitate poate furniza mai multe servicii; prin urmare, poate participa la mai multe cazuri de utilizare (în reprezentarea grafică, acestea sunt reunite într-un dreptunghi).

Între cazurile de utilizare ale aceleiaşi entităţi pot exista trei tipuri de relaţii:

- asocierea, pentru reprezentarea participării unui actor la un caz de utilizare;
- extensia, care permite adăugarea de cazuri de utilizare pe parcursul dezvoltării sistemului (cuvântul rezervat „extend" în engleză, se plasează pe săgeata care indică legătura); o astfel de relaţie presupune o condiţie de autorizare şi referinţe spre unul sau mai multe puncte de extensie din cazurile de utilizare destinaţie;
- incluziunea, care semnalează faptul că o instanţă a unui caz de utilizare furnizează servicii specificate printr-un alt caz de utilizare (cuvânt rezervat în engleză: „include");
- generalizarea: o relaţie de generalizare de la cazul A la cazul B se defineşte cu ajutorul relaţiei complementare de specializare şi specifică faptul că B este o specializare a lui A (simbolul grafic este cel clasic, prezentat în figura 1).

Acest tip de relaţie dintre cazurile de utilizare conduce la o generalizare şi între actori. Astfel, dacă actorul A1 este o specializare a lui A, atunci acesta din urmă poate participa la realizări ale cazurilor de utilizare corespunzătoare lui A1.

În UML, un subsistem desemnează un grup de elemente de modelare, care modelează o unitate comportamentală a sistemului.

[3] Datorită acestui „repository", orice schimbare intervenită în vreuna dintre diagramele modelului se va opera automat şi în celelalte diagrame ale acestuia.

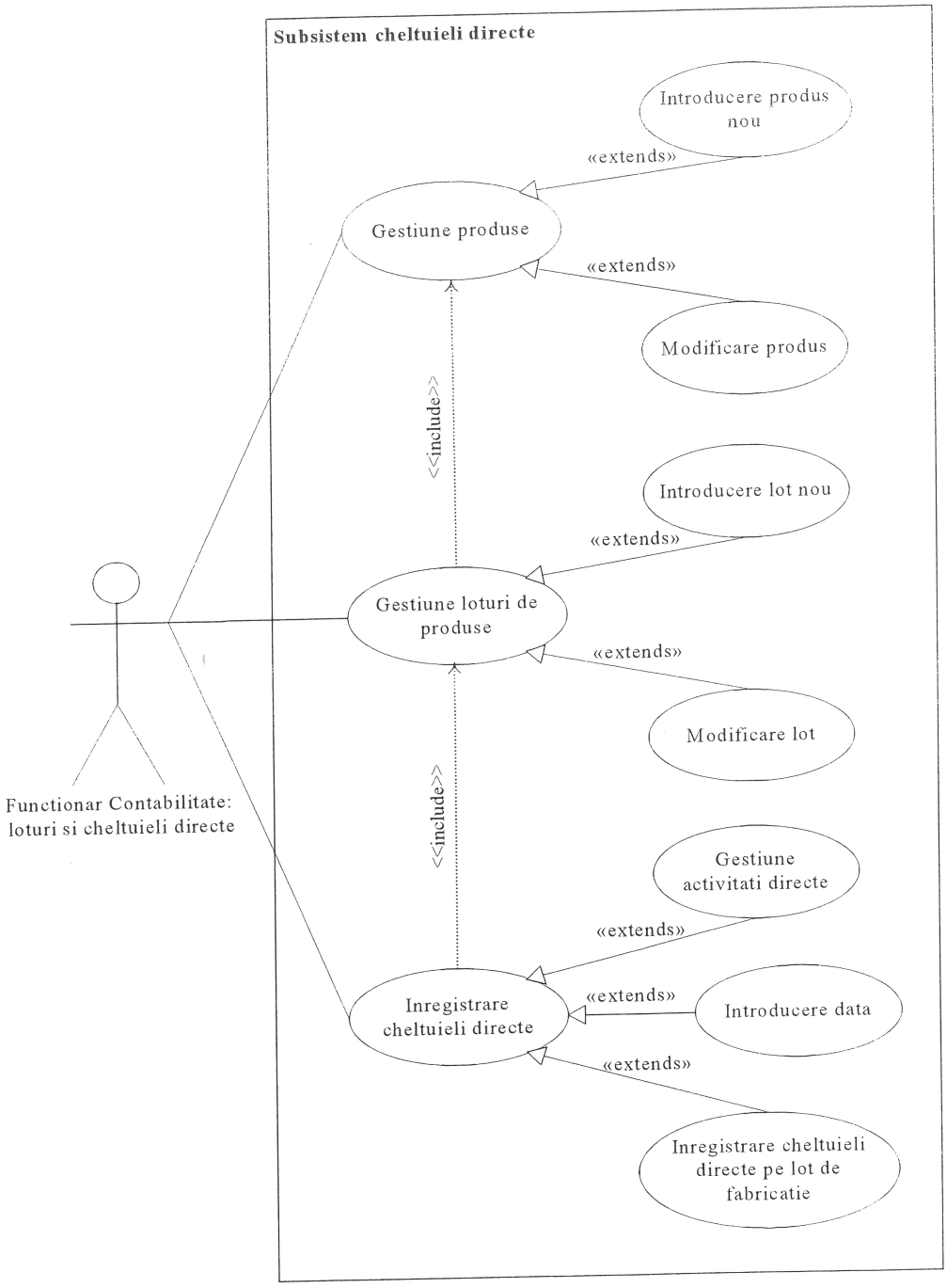

Figura 2. Cazuri de utilizare pentru gestiunea cheltuielilor directe

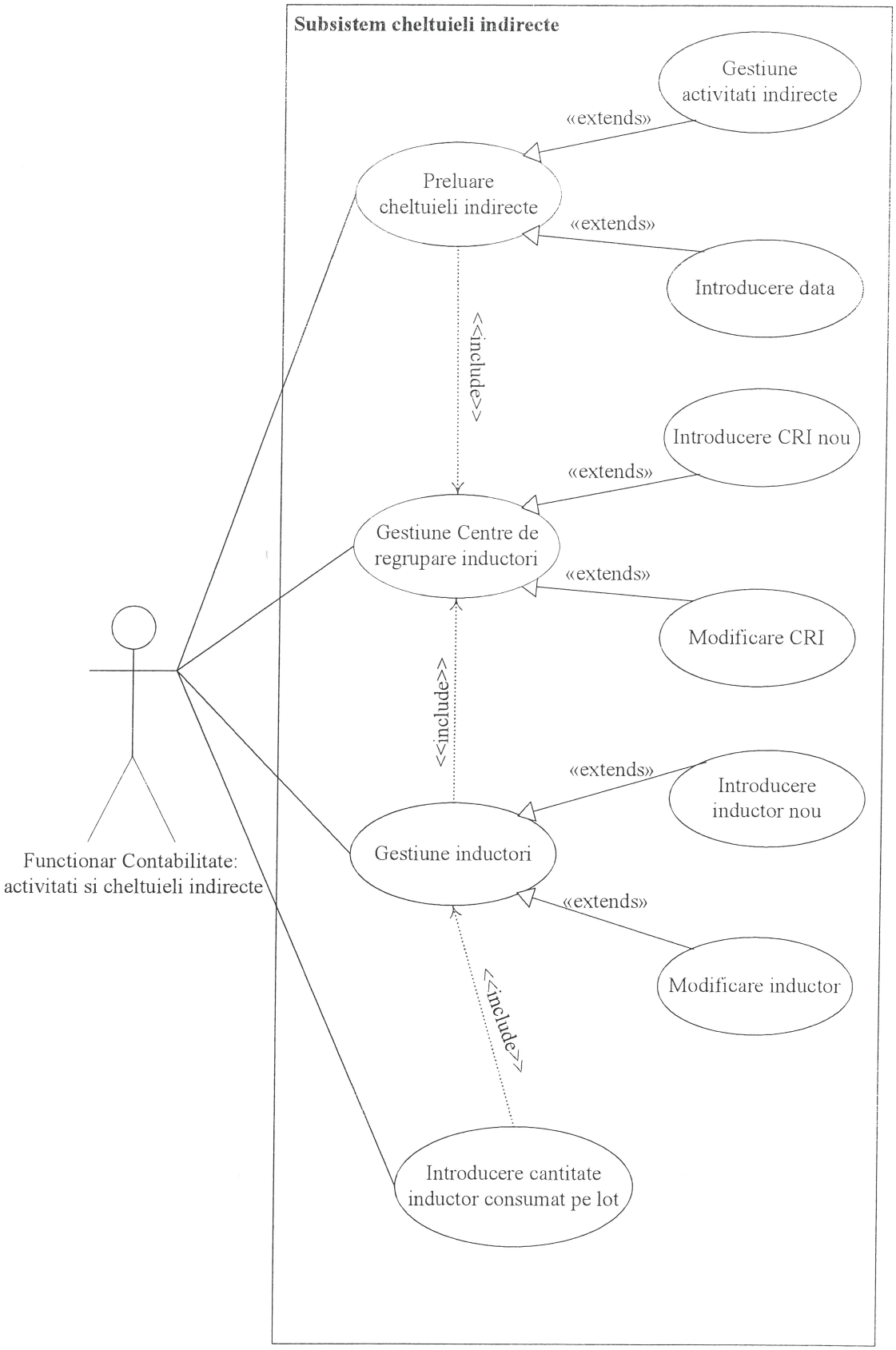

Figura 3. Cazuri de utilizare pentru gestiunea cheltuielilor indirecte

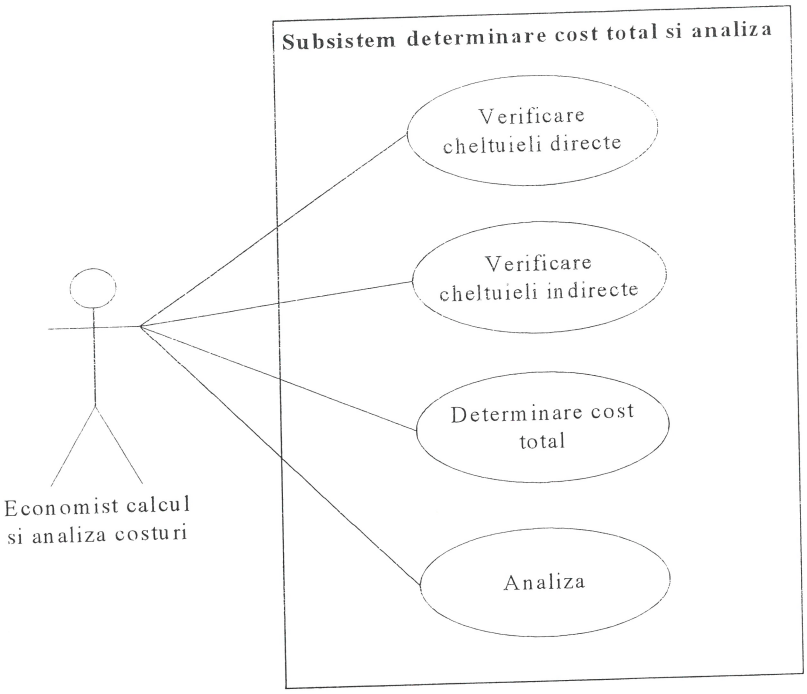

Figura 4. Cazuri de utilizare pentru determinarea şi analiza costului total

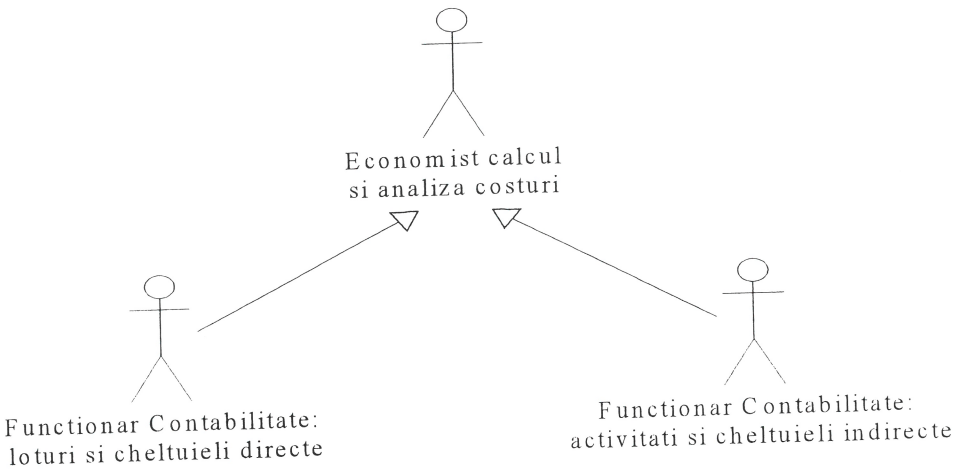

Figura 5. Conform relaţiei de generalizare, actorul „Economist" poate realiza şi funcţiunile celorlalţi doi actori implicaţi în cazurile de utilizare prezentate

Cazurile de utilizare prezentate oferă o viziune exterioară a sistemului. Din interior, realizarea cazurilor de utilizare se poate analiza din următoarele perspective:

- structurală (diagrama claselor);
- spaţială (diagrama de colaborări);
- temporală (diagrama de secvenţe);
- a acţiunilor (diagrama de activităţi).

3. Diagrama claselor

În modelarea structurii claselor şi obiectelor se urmăreşte realizarea diagramelor claselor şi a obiectelor.

O clasă este unul dintre aceste elemente de modelare şi reprezintă „o extindere a conceptului de entitate, cu operaţii"[4]. Gradul de încapsulare a atributelor şi metodelor este indicat de vizibilitate - proprietate a acestora care

[4] [8]. p. 667.

stabileşte dacă pot fi invocate în afara spaţiului în care au fost declarate şi definite. Specificaţiile UML fac distincţie „între atributele private, precedate de – şi atributele publice notate cu +".[5] Există şi notaţia # pentru calificativul de protejat.

UML furnizează instrumentele necesare reprezentării diagramelor de stare şi de colaborare; definirea claselor de obiecte conduce imediat la descoperirea legăturilor de specializare şi de generalizare dintre clase. În reprezentarea moştenirilor, UML permite notarea explicită a legăturilor de tip exclusiv ori inclusiv, prin notarea restricţiilor {Exclusive}, respectiv {Inclusiv} pe model, acolo unde este cazul. Pentru a arăta că o clasă este compusă din una sau mai multe subclase, se utilizează agregarea, independentă (o asociere care leagă un obiect cu unul sau mai multe obiecte componente, clasele fiind autonome; simbol: ◊) ori compozită (ilustrează cazul obiectelor rezultate din agregarea de valori; simbol: ◆).

O asociere între clase de obiecte se caracterizează prin:

- multiplicitate: se defineşte ca număr de instanţe ale unei clase date, care corespunde unui obiect al clasei (claselor) asociate. Sunt utilizate notaţiile:
  - 1 – o singură instanţă a clasei date (un singur obiect);
  - 0..1 – zero sau o instanţă;
  - * - zero sau mai multe instanţe;
  - 1..* - una sau mai multe instanţe;
  - 2..5 – minim două, maxim cinci instanţe.
- ordonare: se precizează numai pentru multiplicităţi mai mari decât 1, indicând dacă obiectele trebuie păstrate într-o anumită secvenţă de ordonare;
- navigabilitate: stabileşte sensul de parcurgere al asocierii;
- rol: desemnează rolul deţinut de o clasă dată într-o anumită asociere;
- vizibilitate: de obicei, dacă se utilizează, precede numele de rol;
- posibilităţi de modificare:
  - implicit – orice modificare va fi permisă;
  - „frozen" – după crearea asocierii nu va mai fi posibilă nici o modificare;
  - „AddOnly" – ulterior creării asocierii, vor fi permise numai adăugări (nu şi modificări ori ştergeri).

Diagrama claselor la nivel conceptual, propusă pentru un model orientat obiect al sistemului ABC de gestiune şi analiză a costurilor, este prezentată în figura 6.

Pentru detalierea aspectelor legate de o eventuală implementare obiectuală a sistemului, diagrama conceptuală a claselor se poate transforma într-o diagramă la nivel de implementare; pentru fiecare clasă se specifică toate atributele şi operaţiile, stabilindu-se vizibilitatea fiecărei componente. De asemenea, se precizează tipuri de date specifice limbajului de implementare („byte", „string"), şi nu tipuri de date generice („numeric", „text"). Diferitele tipuri de relaţii între clase (agregare, compoziţie sau moştenire) pot apărea în această diagramă numai dacă sunt suportate de limbajul în care se face implementarea; de asemenea, trebuie să se specifice modalităţile de asigurare a persistenţei obiectelor aplicaţiei.

[5] Idem 4;

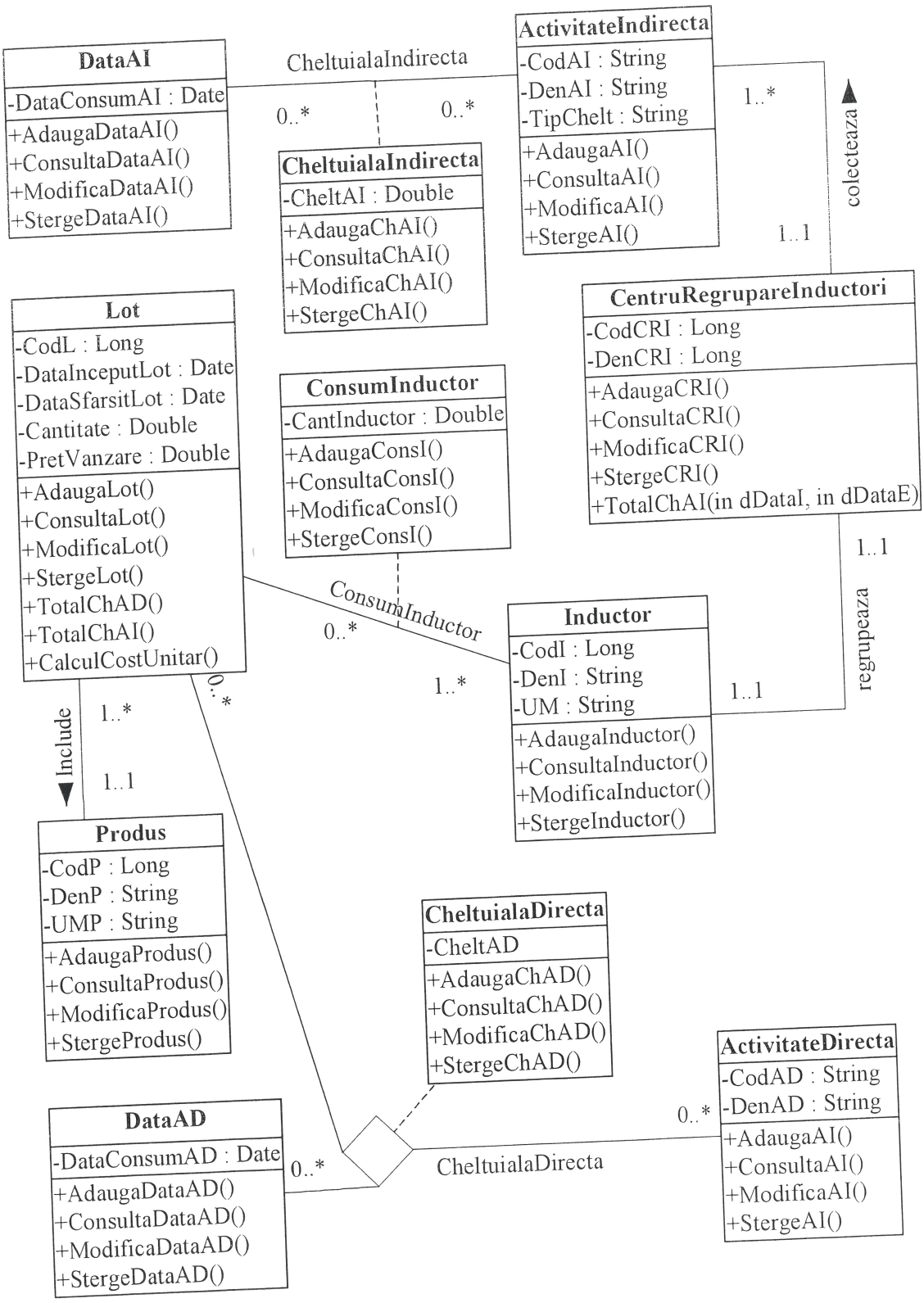

Figura 6. Diagrama claselor (nivel conceptual)

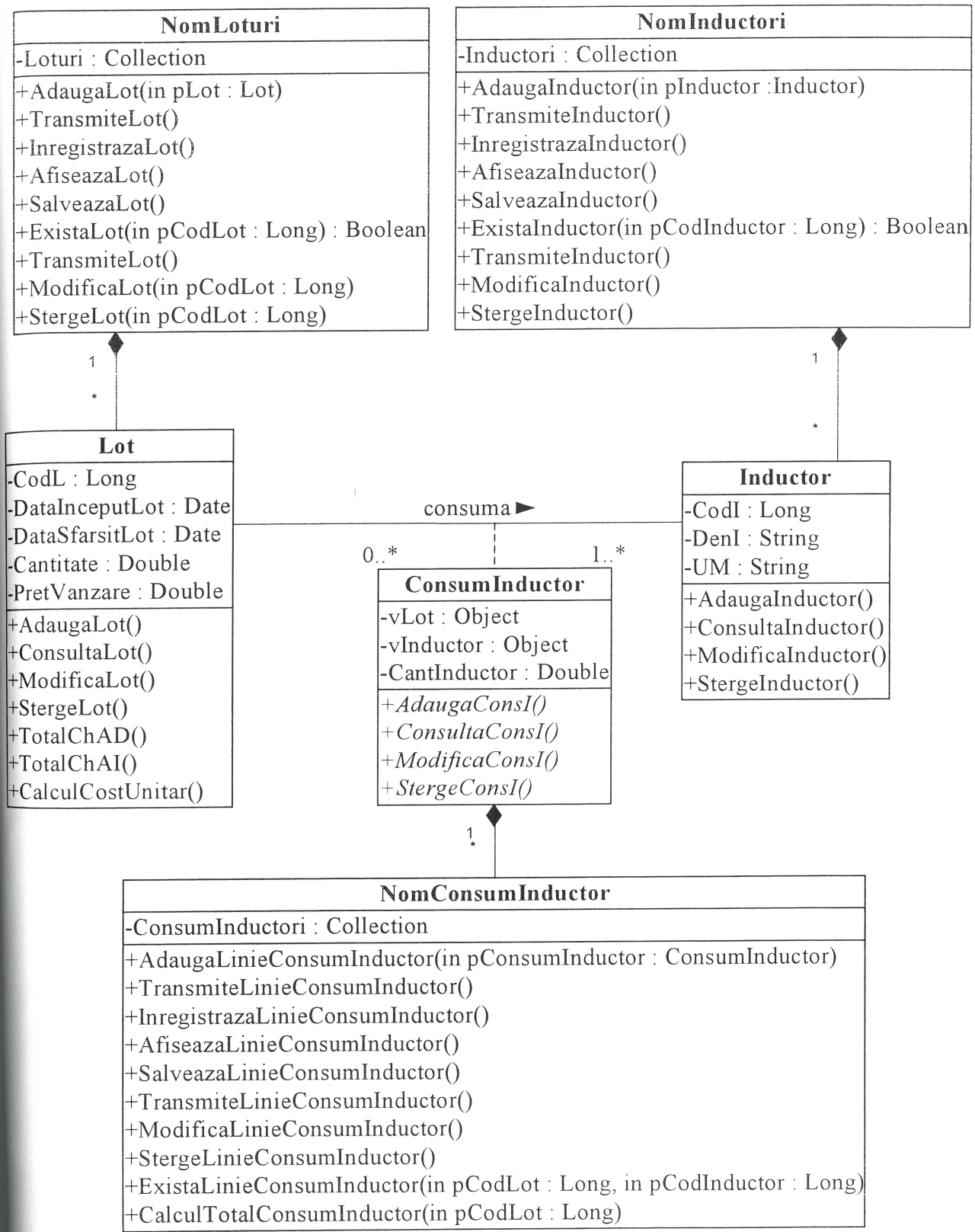

Figura 7. O parte de diagramă a claselor (la nivel de implementare)

Bibliografie

1. **ANICA-POPA, I., L. ANICA-POPA, L.**: Programarea calculatoarelor în limbajul Visual Basic. Teorie şi aplicaţii, Editura Infomega, Bucureşti, 2002.
2. **AREITIO, M. DEL C.**: Análisis de sistemas: fundamento y desarrollo de aplicaciones informáticas, Editura Areber, Bilbao, 1992.
3. **AREITIO, M.DEL C.**: Ingeniería del Software. Guía de estudio, Editura G.C., Bilbao, 1995.
4. **CAGWIN, D., M. J. BOUWMAN**: The Association Between Activity-Based Costing And Improvement In Financial Performance, aprilie 2000.
5. **DE AMESCUA SECO, A.**: Ingeniería del software de gestión: análisis y diseño de aplicaciones, Editura Paraninfo, Madrid, 1995.
6. **DRUCKER, P.F.**: The Information Executives Truly Need. În: Harvard Business Review, ianuarie/februarie, 1995.
7. **EMBLEMSVAG, J.**: Historic Overview of Modern Management Practices. În: SURF Study Trip, 2000.
8. **GARDARIN, G.**: Bases de données: object et relationnel, Maison d'Edition Eyrolles, Paris, 2001.
9. **GRANOF, M., D. PLATT, I. VAYSMAN**: Using Activity-Based Costing to manage more effectively, The Pricewaterhouse Coopers Endowment for The Business of Government, ianuarie, 2000.
10. **Grupul BDASEIG, V. FLORESCU** (coordonator): Baze de date. Fundamente teoretice şi practice, Editura Infomega, Bucureşti, 2002.
11. **LEÓN SERRANU, G.**: Ingeniería de sistemas de software, Editura Isdefe, Madrid, 1996.
12. **OPREA, D.**: Analiza şi proiectarea sistemelor informaţionale economice, Editura Polirom, Iaşi, 1999.
13. **POPOVICI, D.M., I. M. POPOVICI, J. G. RICAN**: Proiectare şi implementare software, Editura Teora, Bucureşti, 1998.
14. **RAVIGNON, L., P.-L. BESCOS**: Gestion par activités. La méthode ABC/ABM. Piloter efficacement une PME, Edition d'Organisation, Paris, 1998.
15. **SARU, D., A. D. IONIŢĂ**: Dezvoltarea orientată pe obiecte a programelor medii şi mari în limbajul C++", Editura Matrix Rom, Bucureşti, 1998.
16. **WHITTEN, J., I. BENTLEY, V. BARLOW**: Análisis de sistemas y métodos de diseño: édicion para el instructor", traducción, Diorki, ed. Irwin, Barcelona, 1996.
17. **ZAHARIE, D., I. ROŞCA, I.**: Proiectarea obiectuală a sistemelor informatice, Editura Dual Tech, Bucureşti, 2002.