UML- Limbaj unificat de modelare

4.1. Introducere în UML

Limbajul unificat de modelare, numit prescurtat UML se doreşte a fi un instrument care să ofere o metodologie unificată de modelare, analiză şi proiectare a sistemelor informatice.

El oferă o arhitectură de sistem pentru analiza şi proiectarea orientată obiect a sistemelor software, incluzând şi documentarea acestora, dar şi pentru modelarea altor sisteme non-software, cum ar fi modelarea afacerilor, de exemplu.

- Metode de proiectare orientate pe obiect
- Metode de proiectare structurate

} ⇨ Modele inconsistente

Scopurile principale sunt reutilizarea, portabilitatea şi interoperabilitatea software-lui orientat pe obiect.

Adoptarea specificaţiei UML a redus gradul de confuzie în cadrul industriei limbajelor de programare şi a permis schimbul reciproc între instrumentele vizuale de dezvoltare.

Prima sa versiune a apărut în Ianuarie 1997 şi a fost dezvoltat: Grady Booch, Jim Rumbaugh şi Ivar Jacobson.

UML a fost selectat ca standard al limbajelor de modelare orientate obiect (OOML- Object Oriented Modeling Language) de către OMG şi este utilizat de dezvoltatorii de produse software.

CE ESTE UML ?

Sunt multe definiţii şi nici una unanim acceptată sau standardizată.

Dar toate au în comun următoarele:

UML este un limbaj grafic pentru vizualizarea, specificarea, construrirea şi documentarea componentelor unui sistem software.

Mai mult se poate spune **că UML este un limbaj vizual de modelare, dar care nu are pretenţia de a fi un limbaj de programare vizual, el putând fi utilizat ca un limbaj universal pentru descrierea sistemelor**.

UML poate fi utilizat in toate domeniile ingineriei software oferind un mod standard de a scrie proiecte de sistem, incluzând obiectele conceptuale şi funcţiile de sistem precum şi obiecte concrete cum ar fi declaraţiile din limbajul de programare, scheme ale bazelor de date şi componente software reutilizabile.

Acest limbaj unificat reprezintă o arhitectură bazată pe patru niveluri de abstractizare definite în metamodelul UMLşi anume:

1. ***Meta-metamodelul*** – infrastructura pentru o arhitectură de modele;
2. ***Metamodelul*** – o instanţă a unui meta-metamodel şi defineşte semantica necesară pentru reprezentarea modelelor aplicaţiei;
3. ***Modelul*** – o instanţă a unui metamodel;
4. ***Obiecte utilizator*** – o instanţă a unui model, utilizate pentru descrierea unui domeniu specific de informaţie.

Metamodelul UML este un model logic şi are în componenţa sa trei pachete logice:

- *Elemente de comportament (Behavioral Elements);*
- *Elemente de bază (Foundation);*
- *Mecanisme generale (Model Management).*

Avantajele unui metamodel logic este acela că accentuează declarativele semantice, înlăturând detalile de implementare.

Implementările care utilizează metamodelul logic trebuie să se conformeze semanticilor sale şi să fie capabil să importe şi să exporte obiecte.

În cadrul fiecărui pachet, elementele unui element sunt definite astfel: sintaxă abstractă, reguli foarte bine formulate sau reguli de corectitudine şi semantică.

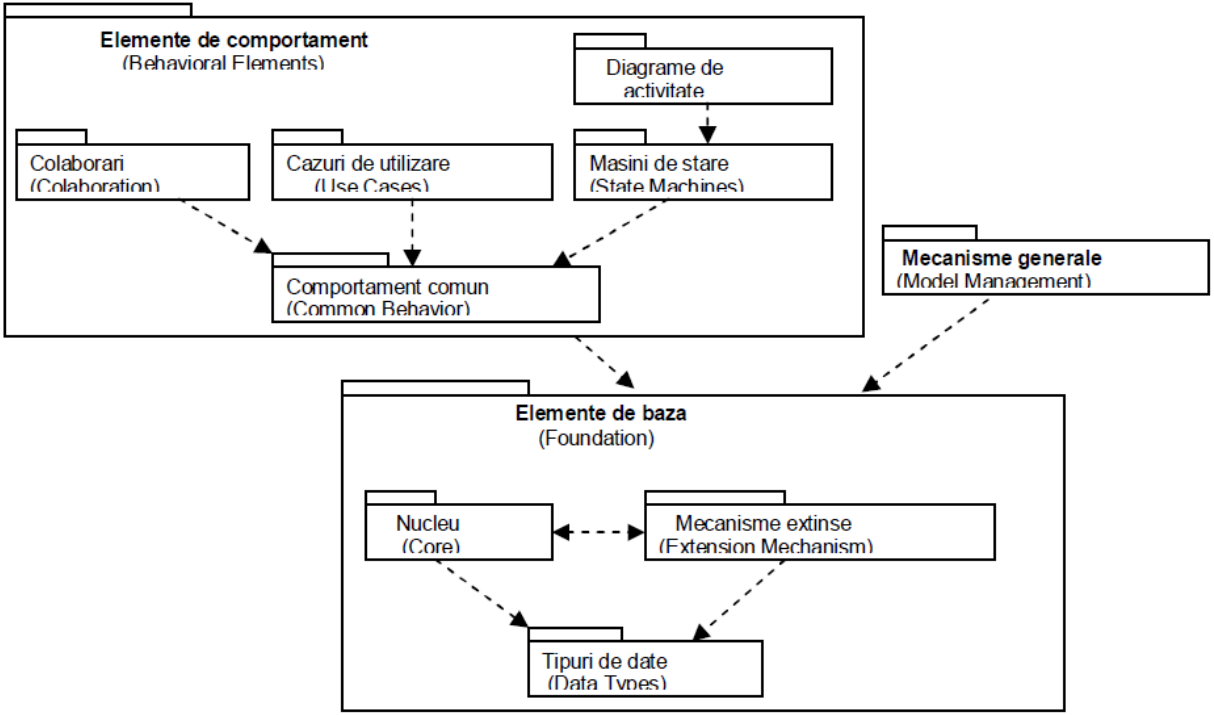


Fig. 4.1. Structura UML

Componentele de bază ale UML sunt elementele model.

Un ***model*** reprezintă o colecţie de *obiecte* (clase, pachete, actori, cazuri de utilizare, componente şi noduri), *relaţii* (de asociere, dependenţă, generalizări) şi *diagrame*.

Un model :

- Este entitatea de baza a proiectarii;
- Este individualizat;
- Este legat de alte modele prin legaturi;
- Este reprezentat grafic.

Modelele sunt de două tipuri:

- ***modele structurale*** *(statice)* care accentuează structura obiectelor din sistem, incluzând şi clasele lor, interfeţele, atributele şi relaţiile şi
- ***modele de comportament*** *(dinamice)*, care accentuează comportamentul obiectelor din sistem, incluzând şi metodele, interacţiunea, colaborările şi starea lor istorică.

Principalele obiective ale UML-ului sunt:

- Să pună la dispoziţia utilizatorului un limbaj vizual pentru dezvoltarea şi specificarea sistemului;
- Să suporte concepte de dezvoltare de nivel superior cum sunt componente, colaborări, modele;
- Să încurajeze utilizarea limbajelor de programare orientate pe obiect;
- Să furnizeze o bază formală pentru înţelegerea limbajului de modelare.

4.2. Diagrame şi concepte UML

UML defineşte patru tipuri de diagrame :

1. Diagrama de clase (Class Diagram);
   1.1. Diagrama de obiecte (Object Diagram);
2. Diagrama cazurilor de utilizare (Use Case Diagram);
3. Diagrame de comportament (Behavior Diagrams);
   3.3.1. Diagrama de stare, sau grafice de stări (Statechart Diagram);
   3.3.2. Diagrama de activitate (Activity Diagram);

3.3.3. Diagrama de interacţiuni (Interaction Diagrams);

3.3.4. Diagrama secvenţială (Sequence Diagram);

3.3.5. Diagrama de colaborare (Collaboration Diagram);

4. Diagrame de implementare (Implementation Diagrams);

4.1. Diagrama de componente (Component Diagram);

4.2. Diagrama de aplicaţie (de dezvoltare) (Deployment Diagram).

Aceste diagrame furnizează perspective multiple şi diferite asupra sistemului din punctul de vedere al analizei şi/sau dezvoltării.

4.2.1. Diagrama de clase (Class Diagram)

O diagrama a claselor este un graf ale cărui noduri sunt clasele din sistem şi ale cărui arce sunt relaţiile dintre clase.

Diagrama claselor este probabil cea mai importantă diagramă UML.

O astfel de diagramă poate conţine interfeţe, pachete, legături, şi chiar instanţe, cum ar fi obiecte. Clasele sunt tipuri abstracte de date. În cadrul programului se lucrează cu instanţieri ale claselor definite, numite *obiecte*.

Clasele de obiecte sunt categorii de obiecte cu **atribute** (structuri de date) şi **operaţii** (comportament) comune.

Fiecare obiect are propria sa:

- **stare** (definita de un set de valori păstrate în atributele sale);
- **identitate** (obiectele se disting între ele prin existenţă , nu prin atribute);
- **comportament** (felul în care un obiect acţionează şi reacţionează în termenii comunicării prin mesaje şi schimbării ale stării).

Scopul diagramei de clase este de a prezenta structura de clase din cadrul unui model.

În aplicaţiile orientate pe obiect clasele au atribute, operaţii şi relaţii cu alte clase.

Elementul fundamental al diagramei claselor este un dreptunghi care reprezintă o clasă, aşa cum este ilustrată în figura 4.2.

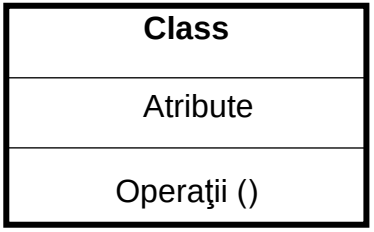


Fig. 4.2. Reprezentarea unei clase din diagrama de clase UML

Dreptunghiul care reprezintă clasa este împărţit în trei compartimente cu următoarele semnificaţii:

- Compartimentul de sus este cel mai important şi conţine numele clasei;
- Compartimentul din mijloc conţine o listă de atribute;
- Compartimentul de jos conţine o listă de operaţii.

În multe diagrame compartimentele de mijloc şi de jos sunt omise. Chiar şi atunci când sunt prezente, ele nu arată toate atributele şi operaţiile. Scopul este de a arata doar acele atribute şi operaţii care sunt esenţiale pentru diagrama respectivă.

Diagrama claselor poate conţine următoarele elemente model:

-  **Pachete.** Pachetele sunt utilizate pentru a structura modelul.
-  **Dependenţe între pachete.** Acestea arată că respectivele clase dintr-un pachet utilizează clasele pachetului de care depind.
-  **Colaborări între obiecte.**
-  **Interfeţe.** Interfeţele nu conţin atribute, ci doar operaţii.

- **Clase.** Clasele reprezintă cel mai important concept al programării orientată pe obiect şi al UML.
- **Relaţii de moştenire.** Acestea se pot regăsi între interfeţe sau între clase, dar niciodată între o interfaţă şi o clasă.
- **Relaţii de implementare.** Pot exista numai între interfeţe şi clase.
- **Relaţii de asociere.** Asocierile sunt relaţii între clase.

În construirea claselor care vor sta la baza proiectării programului trebuie clarificate entităţile care vor evolua în sistem şi operaţiile asociate.

Trebuie de asemenea definite responsabilităţile fiecărei clase în implementare şi colaborările între clasele modelate grafic prin legături de diverse tipuri şi cu diferite multiplicităţi.

Legăturile sunt simbolul relaţiilor între clase.

Un exemplu de reprezentare a unei clase şi anume clasa *Cerc* este prezentat în figura 4.3.

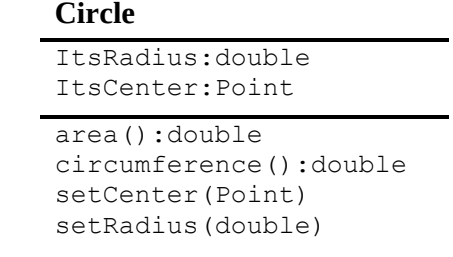


Fig. 4.3. Reprezentarea clasei *Circle*

Fiecare instanţă de tip *Circle* pare că are o instanţă de tip *Point.*

Aceasta este o relaţie cunoscută ca *relaţie de compoziţie* şi figurată în UML ca în fig. 4.4.

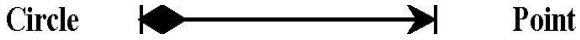


Fig. 4.4. Relaţie de compoziţie

Rombul negru reprezintă compoziţia. El este plasat în dreptul cercului deoarece cercul este compus din puncte. Punctul nu trebuie să ştie nimic despre cerc. Săgeata din partea cealaltă denotă faptul că relaţia poate fi parcursă numai într-un sens.

În UML se presupune că relaţiile sunt implicit bidirecţionale cu excepţia când se termină printr-o săgeată aşa cum am prezentat mai sus.

Dacă s-ar omite săgeta ar însemna că Punctul trebuie să cunoască Cercul, ceea ce la nivel de cod ar însemna să se include # *include " Cercle.h"* în *Point.h*.

Relaţiile de compoziţie sunt mai puternice decât cele de conţinut sau agregare.

Agregarea este întregul sau o parte din relaţie.

În cazul prezentat *Cercle* este întregul iar *Point* parte a *Cercle.*

Relaţia de compoziţie indică de asemenea că timpul de viaţă al *Point* depinde de *Cercle*. Aceasta înseamnă că, dacă Cercul este distrus va fi distrus şi Punctul.

În limbajul C++ această clasă se reprezintă astfel:

```
Class Circle {
   public:
    void SetCenter(const Point&);
    void SetRadius(double);
    double Area() const;
    double Circumference() const;
 private:
     double itsRadius;
   Point itsCenter;
};
```

În acest caz s-a reprezentat relaţia de compoziţie ca o variabilă membră. Se poate folosi la fel de bine un pointer care la sfârşit v-a fi şters de destructorul Cercului.

Clasificatori

Un *clasificator* este un element care descrie caracteristicile de comportament şi structurale ale sistemului.

Clasificatorii acceptaţi de UML sunt de mai multe tipuri şi includ: clase, tipuri de date, interfeţe, componente, semnale, noduri, cazuri de utilizare şi subsisteme.

Un clasificator declară un set de caracteristici care includ atribute, metode şi operaţii.

Numele unui clasificator este unic şi reprezintă o metaclasă abstractă.

UML admite următoarele tipuri speciale de clasificatori, numite *stereotipuri* :

- <<metaclasă>> - precizează că instanţele clasificatorului sunt clase;
- <<powertype>> - specifică un clasificator ale cărui obiecte sunt descendenţii unui anumit părinte;
- <<process>> – un clasificator care reprezintă un flux de control cu o interfaţă puternică;
- <<thread>> – un clasificator care reprezintă un flux de control;
- <<utility>> – specifică un clasificator care nu are instanţe;

Un *atribut* descrie un domeniu de valori pe care le pot lua instanţele unui clasificator.

El poate avea o valoare iniţială şi una dintre următoarele proprietăţi care se referă la posibilităţile de modificare a valorii, după ce obiectul a fost creat:

- *changeable* (modificabil)- nu se impune nici o restricţie asupra valorii atributului;

- *addOnly* – valabil numai pentru atributele cu ordin de multiplicitate mai mare ca unu; o valoare odată adăugată nu mai poate fi ştearsă sau modificată;
- *frozen* – valoarea atributului nu poate fi modificată după iniţializarea obiectului.

O **operaţie** este un serviciu care poate fi solicitat de către un obiect.

O operaţie în UML poate avea una dintre următoarele proprietăţi care se referă la simultaneitatea apelurilor concurente către o clasă pasivă:

- *isQuery* – indică dacă se schimbă sau nu starea sistemului. Dacă are valoarea *true* starea sistemului rămâne neschimbată.
- *sequential* – apelurile trebuie efectuate secvenţial;
- *guarded* – se pot produce simultan apeluri multiple către o instanţă dar numai unul are permisiunea să înceapă, celelalte fiind blocate până la finalizarea operaţiei.
- *concurrent* - se pot produce simultan apeluri multiple către o instanţă, fiecare dintre ele putând fi realizate în paralel.

Operaţiile pot avea parametri care sunt utilizaţi pentru specificare şi fiecare include un nume, un tip şi direcţia comunicării.

Direcţia se poate indica astfel:

- *in* – parametru de intrare care nu poate fi modificat;
- *out* – parametru de ieşire care poate fi modificat pentru a transmite informaţii către alte elemente;
- *inout* – parametru de intrare care poate fi modificat;
- *return* – valoare returnată de un apel.

Multiplicitatea unei clase specifică numărul de instanţe pe care le poate avea. Multiplicitatea se aplică şi la nivel de atribut.

O diagramă tipică de clase este prezentată în figura 4.5.

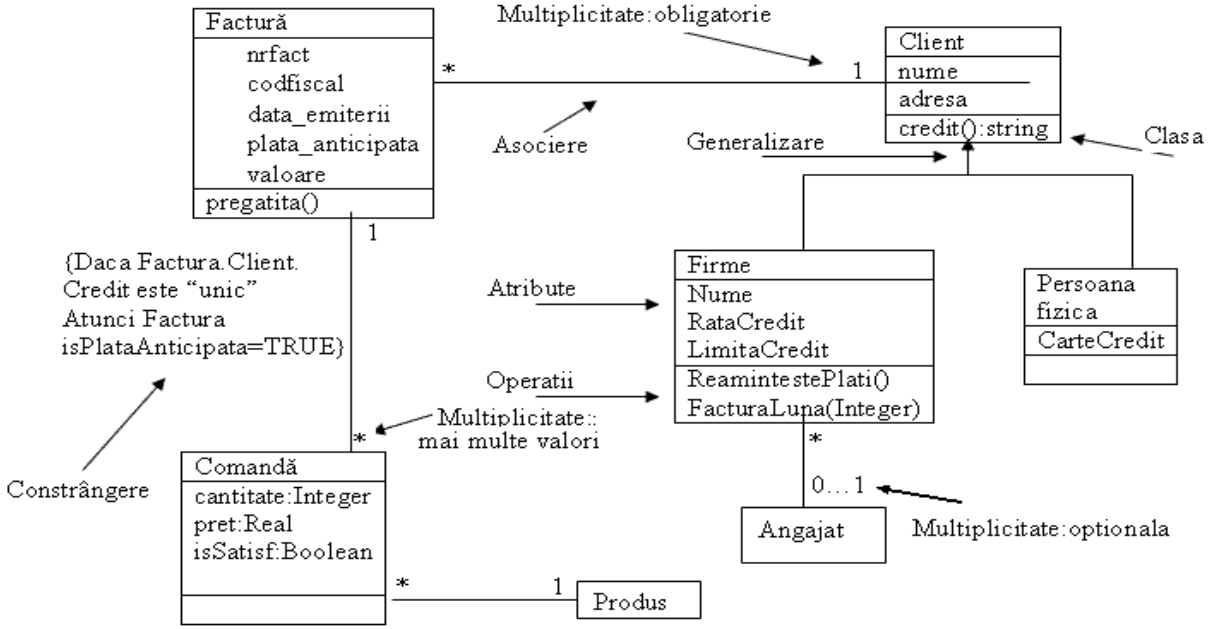


Fig. 4.5. Diagramă de clase

4.2.1.1. Diagrama de obiecte

Această diagramă evidenţiază un set de obiecte şi relaţiile cu alte obiecte la un moment dat.

O diagrama de obiecte poate fi considerată un caz special al diagramelor claselor sau al diagramei de colaborări şi este o instanţă a unei diagrame de clase.

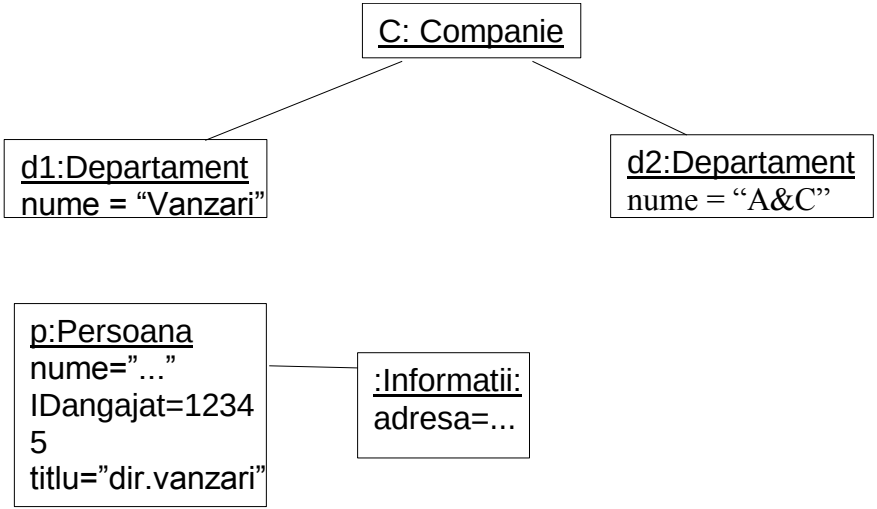
Ea arată o instanţă a unei stări a unui sistem la un moment dat şi conţine: obiecte, legături, eventual note şi constrângeri .

Obiectele pot fi :

- ***Actor*** este un obiect care operează asupra altor obiecte dar asupra căruia nu poate opera alt obiect;
- ***Server*** un obiect care poate opera pe alte obiecte;
- ***Agent*** un obiect care operează asupra altor obiecte şi asupra căruia pot opera alte obiecte. Un agent lucrează în numele unui actor sau altui agent.
- ***Legătura*** reprezintă o instanţă a unei relaţii de asociere şi defineşte o conexiune între instanţe. O legătură trebuie să aibă un furnizor (desemnează elementul care nu este afectat de o modificare) şi un client.
    - Un element furnizor poate participa în mai multe relaţii de legătură către diferiţi clienţi.

- Un element client poate participa numai într-o singură relaţie de legătură cu un furnizor.
- O legătură este o dependenţă unde furnizorul este un model şi clientul reprezintă instanţierea modelului care îndeplineşte substituirea parametrilor modelului;

- ***Note*** pentru formularea constrângerilor şi a altor comentarii sau alte explicaţii referitoare la clasificatorul utilizat;



Această diagramă foloseşte pentru vizualizarea, specificarea, construirea şi documentare structurii obiectelor şi în special pentru a arăta structurile de date.

Modelarea structurii obiectelor presupune:

- Identificarea mecanismului de modelat (o anumită funcţie sau comportamentul unei părţi a sistemului);
- Pentru fiecare mecanism, se identifică clasele, interfeţele şi alte elemente care participă la această colaborare;
- Se consideră un scenariu prin acest mecanism; se determină fiecare obiect care participă la mecanism;
- Selectarea obiectelor care au responsabilităţi de nivel înalt pentru fluxul lucrării;
- Identificarea precondiţiilor stărilor iniţiale şi postcondiţiilor stărilor finale;
- Specificarea activităţilor şi acţiunilor începând cu starea iniţială;
- Evidenţierea tranzacţiilor care conectează aceste activităţi şi acţiuni;
- Evidenţierea obiectelor importante implicate în fluxul de lucrări, cu evidenţierea schimbării valorilor obiectelor.

Modelarea operaţiilor presupune:

- Colectarea abstracţiilor implicate în operaţii (parametri, atribute ale claselor);
- Identificarea precondiţiilor stării iniţiale şi postcondiţiilor stării finale;
- Specificarea activităţilor şi acţiunilor începând cu starea iniţială;
- Folosirea ramificării dacă este necesar;
- Folosirea bifurcării şi reunirii pentru specificarea fluxurilor de control paralele.

4.2.2. Diagrama cazurilor de utilizare

Diagrama prezintă funcţionalitatea sistemului din punctul de vedere al interacţiunilor externe şi este utilizată în etapa de analiză.

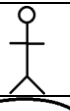
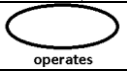
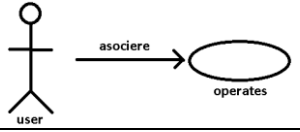
Elementele UML conţinute într-o astfel de diagramă sunt: cazuri de utilizare, actori, relaţii de utilizare (includere), relaţii de extindere, relaţii de generalizare şi de asociere.

Elementele din această diagramă sunt utilizate în primul rând pentru a defini comportamentul sistemului, a subsistemelor, a unei entităţi, fără a specifica structura internă.

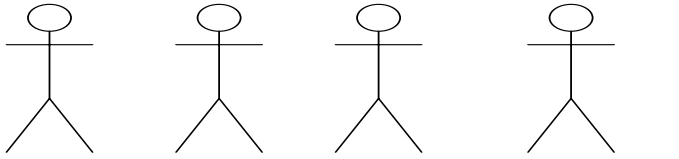
Diagrama cazurilor de utilizare prezintă relaţia dintre actori, cazuri de utilizare şi sistem.

Diagrama este un grafic care conţine actori, un set de cazuri de utilizare, posibile interfeţe şi relaţiile dintre acestea.

1. Componentele diagramei cazurilor de utilizare

Nr.	Denumire componentă	Simbol grafic
1	Actor	
2.	Cazul de utilizare	
3.	Relaţii	

1. Un actor este “cineva” sau “ceva” care interacţionează cu sistemul . Exemple:



Este o entitate internă sistemului (utilizator uman, dispozitiv fizic, un alt sistem), legată de acesta printr-un schimb de informaţie.

Defineşte un set al rolurilor pe care utilizatorii unei entităţi le pot avea în cadrul interacţiunii cu aceasta.

2. Cazuri de utilizare

- Orice caz de utilizare specifică o succesiune de acţiuni (evenimente), cu variantele lor, pe care entitatea le realizează atunci când interacţionează cu actorii săi.
- Comportamentul unui caz de utilizare este specificat prin descrierea setului de acţiuni.

Acces informaţii

Afişează comanda

Afişează cererea

3. Relaţii *Uses* şi *Extends*

Relaţiile de utilizare Uses (includere): Se folosesc atunci când există un comportament care se repetă identic în situaţia mai multor cazuri de utilizare.

Relaţiile de extensie - Extends : Sunt un tip special de generalizare pentru cazurile de utilizare folosite atunci când avem un caz de utilizare similar cu un alt caz, dar care face ceva în plus faţă de acesta.

Exemplul tipic pentru o astfel de diagramă este prezentat în figura 4.6. şi se referă la modelul cumpărării de produse online. Săgeţile pline din figuri reprezintă cazuri “copii” (⟶).

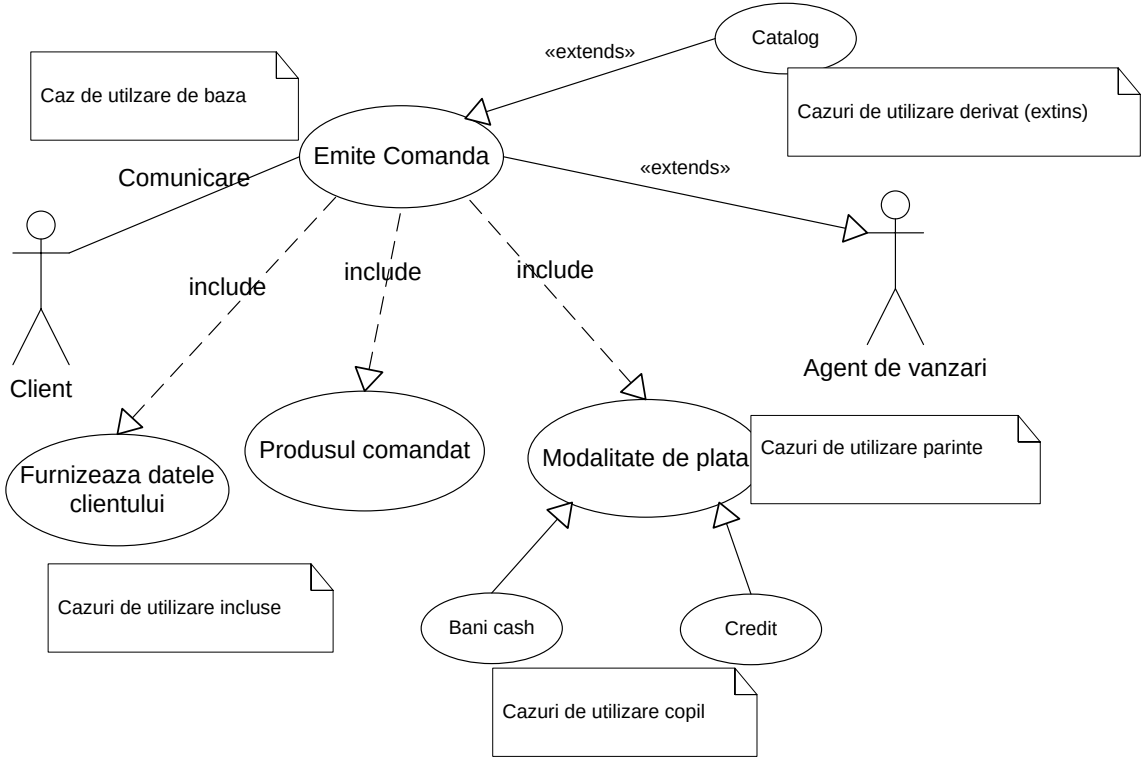


Fig. 4.6. Diagrama cazurilor de utilizare pentru comerţ on-line

Un alt exemplu de diagramă a cazurilor de utilizare este prezentat în figura 4.7. şi se referă la un sistem de evidenţă a studenţilor.

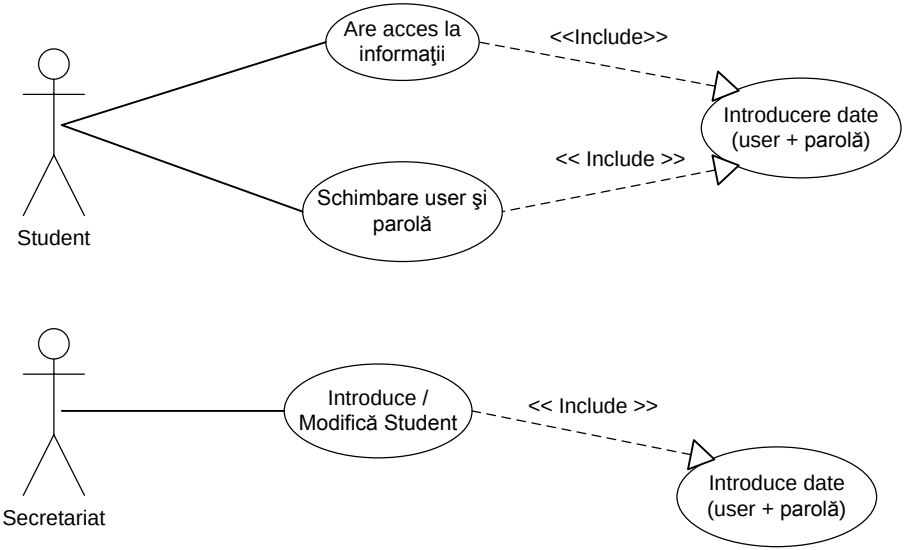


Fig. 4.7. Diagrama cazurilor de utilizare pentru un sistem de evidenţă a studenţilor

Un alt exemplu de diagramă este ilustrat în figura 4.8.

Este prezentat un caz de utilizare al unui sistem destinat urmării activităţii unei agenţii de turism.

Utilizatorul (actor în sistem) se autentifică şi poate deveni client al agenţii.

De asemenea un ofertant de servicii turistice se poate adresa agenţiei şi devine utilizator, dar de alt tip.

Operatorul (angajat al agenţii de turism), actor şi el în sistem, poate vizualiza clienţii existenţi, ofertele şi poate face rezervări.

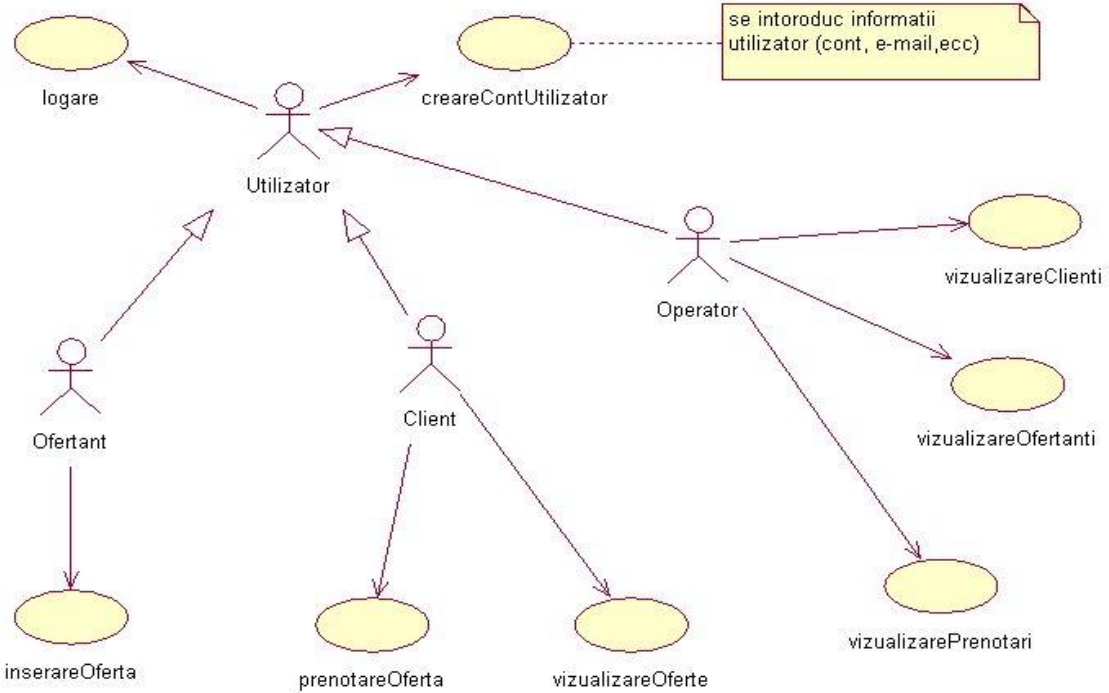


Fig. 4.8. Diagrama cazurilor de utilizare pentru o agenţie de turism

4.2.3. Diagrame de comportament

4.2.3.1. Diagrama de stare

Diagrama de stare modelează comportamentul unui singur obiect (instanţă a unei clase), a unui caz de utilizare, a unui actor sau a întregului sistem arătând de fapt comportamentul orientat - eveniment al obiectului.

O diagramă de stare UML este un graf care poate conţine următoarele elemente: stări, maşini de stări, tranziţii, evenimente, acţiuni şi bare de sincronizare.

O *maşină de stări* este o succesiune de stări prin care trece un obiect, pe durata sa de viaţă ca răspuns la evenimente.

O maşină de stare poate fi reprezentată prin intermediul următoarelor diagrame :

- diagrama de stare, caz în care accentul este pus pe comportamentul ordonat-eveniment al obiectului;
- diagrama de activitate, caz în care accentul este pus pe activităţile ce au loc în obiect.

O *stare* reprezintă o situaţie din viaţa unui obiect, putând satisface anumite condiţii, realizând activităţi sau aşteptând producerea unor evenimente.

O stare poate fi:

- Iniţială, sau de început – arată momentul de iniţiere a maşinii de stare sau al unei substări şi se reprezintă în diagramă printr-un cerc înnegrit;
- Intermediară – o stare prin care trece maşina de stare;
- Finală – maşina de stare a fost executată. Se reprezintă prin două cercuri concentrice, cercul din interior fiind înnegrit.
- Compusă – are în componenţă mai multe substări disjuncte. Se reprezintă printr-un dreptunghi împărţit pe orizontală în două zone.
- Concurenţială. Se reprezintă printr-un dreptunghi împărţit pe orizontală în trei zone.

Într-o diagramă, stările se reprezintă prin dreptunghiuri cu colţurile rotunjite.

Tranziţia reprezintă trecerea de la o stare la alta dintr-o diagramaă de stare. Se reprezintă printr-o săgeată.

Un exemplu de diagramă de stare, cu toate elementele descrise mai sus este prezentată în fig. 4.9 şi reprezintă maşina de stare pentru un obiect *Produs.* Diagrama de stare compusă pentru acelaşi obiect este ilustrată în fig.4.10.

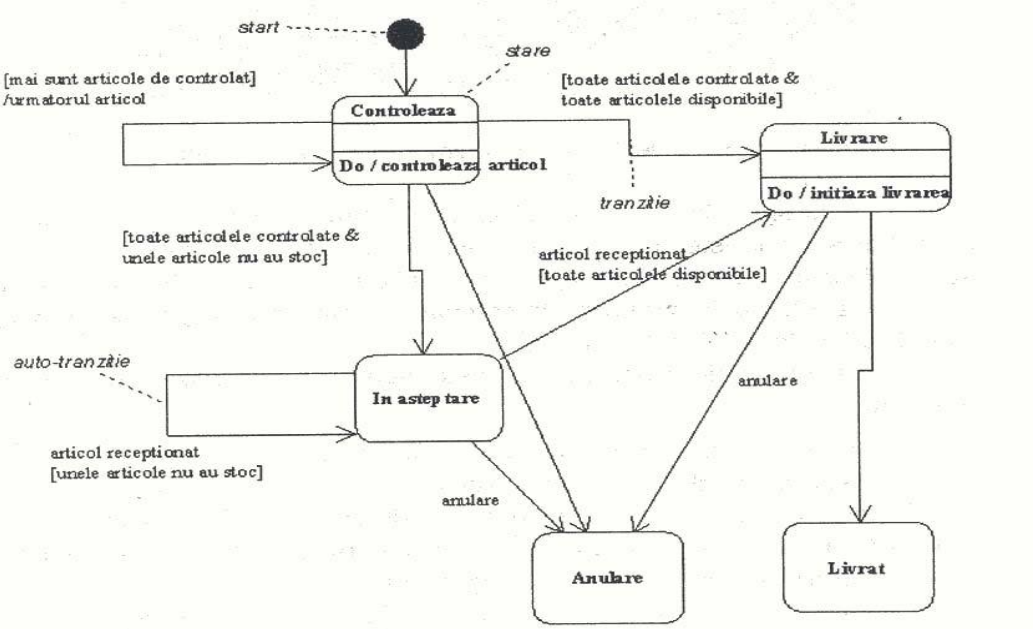


Fig. 4.9. Diagramă de stare

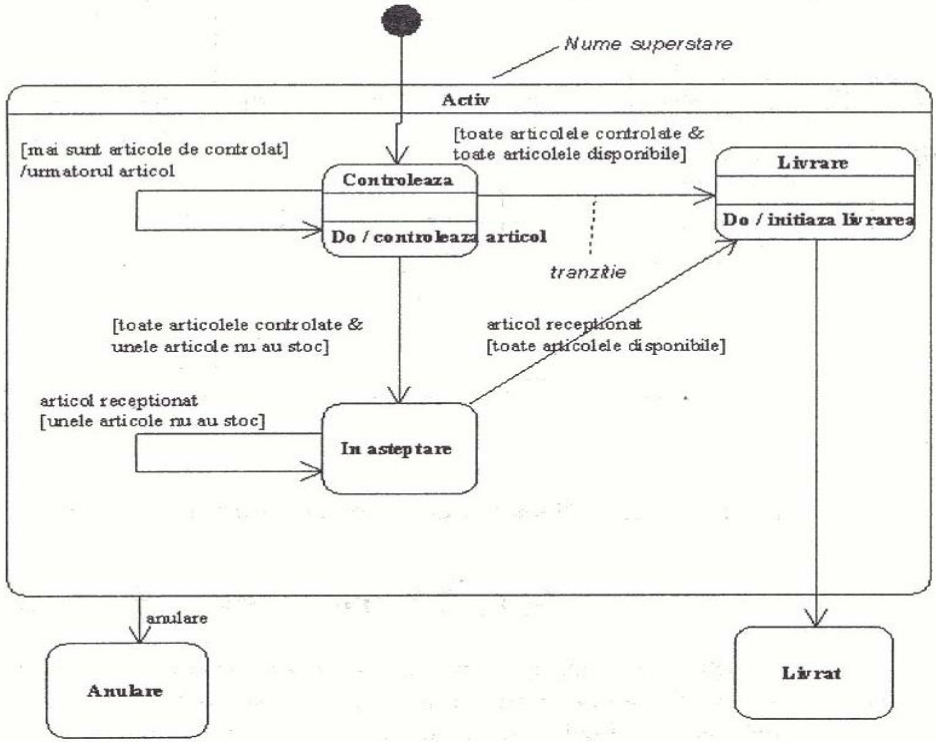


Fig.4.10. Diagrama de stare compusă

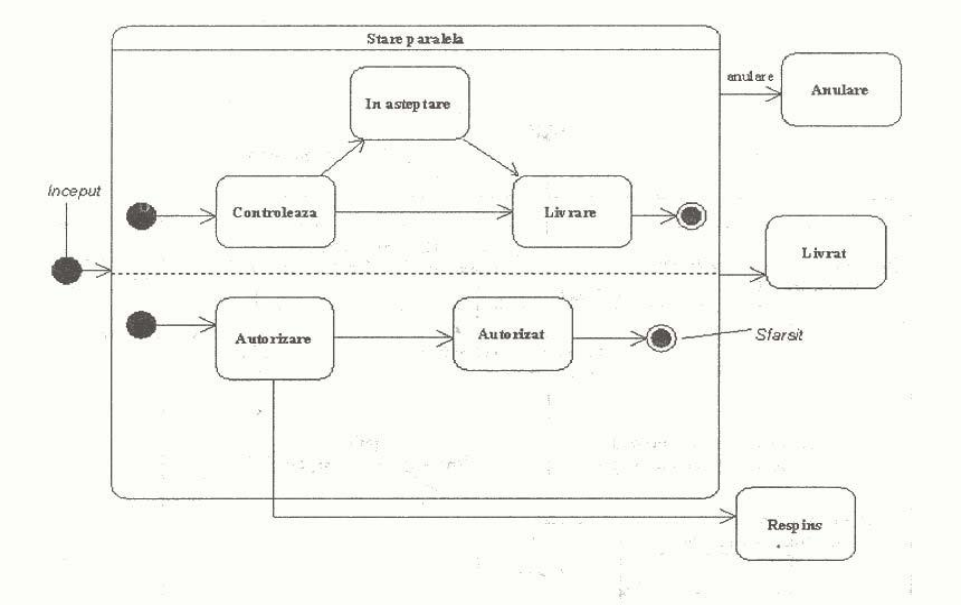


Fig. 4.11. Diagrama de stare concurenţială

Cazul exemplului discutat, agenţia de turism diagrama de stare compusă este ilustrată în figura 4.12.

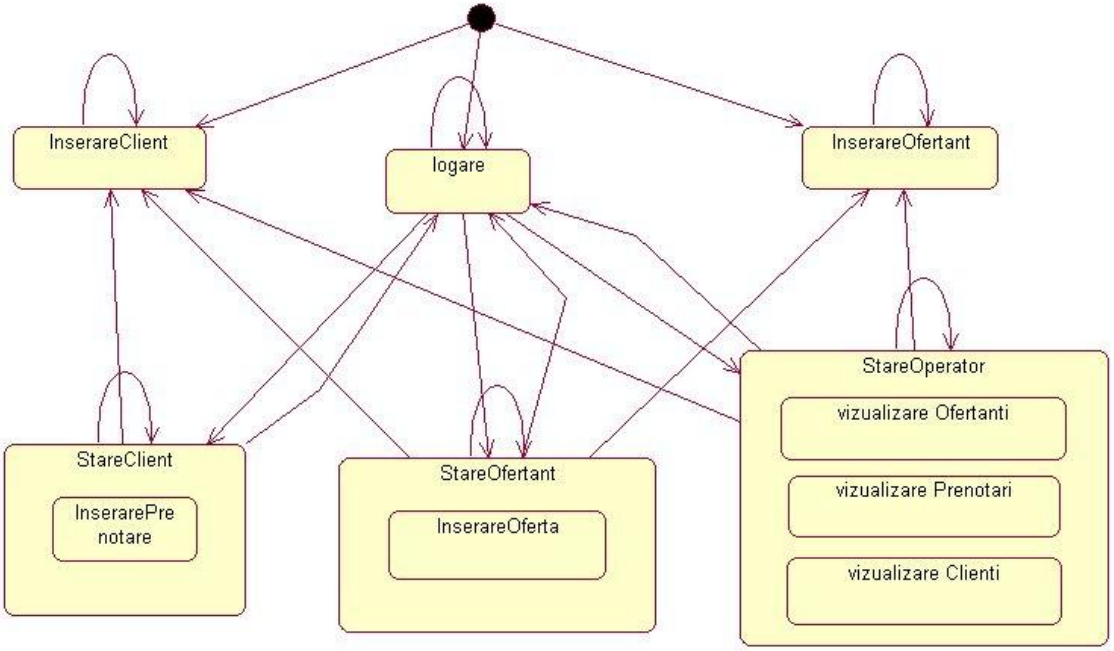


Fig. 4.12. Diagrama de stare compusă pentru agenţia de turism

4.2.3.2. Diagrama de activitate

O diagramă de activitate este un caz particular al diagramelor de stare UML, care defineşte un proces ce evoluează de-a lungul acţiunilor sale.

Această diagramă nu extinde semantica diagramelor UML dar defineşte forme prescurtate care se aplică modelării proceselor.

Este utilizată mai ales atunci când este necesară evidenţierea legăturii fiecărui serviciu sau a mai multor procese paralele.

Diagrama evidenţiază fluzul de control de la o activitate la alta.

O activitate rezultă dintr-o anumită acţiune (apelul unei operaţii, trimiterea unui semnal, crearea sau distrugerea unui obiect), sau evaluarea unei expresii care schimbă starea sistemului sau întoarce o valoare.

Diagrama de activitate este o variantă a unei maşini de stări, în care stările reprezintă performanţa activităţilor sau subactivităţilor.

O stare de activităţi reprezintă o subactivitate care are o anumită durată şi este constituită dintr-un set de acţiuni.

Activităţile reprezintă task-uri ce trebuie realizate de către calculator sau de o persoană, şi vor fi traduse în cadrul modelului prin intermediul unor metode specifice ataşate claselor care vor îngloba comportament de control.

Diagrama de activitate conţine următoarele elemente UML: *stări* (de activitate, de acţiune, de început, de sfârşit), *tranziţii*, *obiecte*, *decizii*, *semnale* (recepţionate sau transmise), *bare de sincronizare*, *culoare* (*swimlane*). Simbolurile lor grafice sunt:


Reprezintă starea iniţială, începutul procesului;


Reprezintă starea finală sau sfârşitul procesului (nu este absolut necesară figurarea stării finale, dacă aceasta reiese clar din contextul activităţilor reprezentate).


Starea de acţiune – stările prin care este posibil să treacă programul în funcţie de ceea ce are de executat.


Bloc de condiţionare ce împarte secvenţa în mai multe alternative


Bloc de bifurcare - este similar conectorului logic "**şi**". Firele de execuţie care pleacă din acest element se pot desfăşura paralel sau pe rând.


Bloc de reunire – element prin care se sincronizează firele de execuţie.

Bloc de sincronizare. Este folosit în cazul subsecvenţelor concurente pentru a sincroniza relaţia “producator-consumator” astfel încât consumatorul să utilizeze resursele disponibile la un moment dat.

Tranziţie –menţine stările active şi elementele modelului împreună.

Reprezintă dependenţe. Se pot trasa între oricare dintre elementele modelului.

Pentru o mai bună înţelegere a diagramelor de activităţi vom considera următorul exemplu:

Se consideră sistemul de gestiune al unei biblioteci. O persoană poate împrumuta sau restitui o carte, nu poate împrumuta nici o carte dacă are datorii către bibliotecă sau dacă are deja cinci cărţi împrumutate. Pentru acest caz diagrama de activitate este prezentată în figura 4.13.

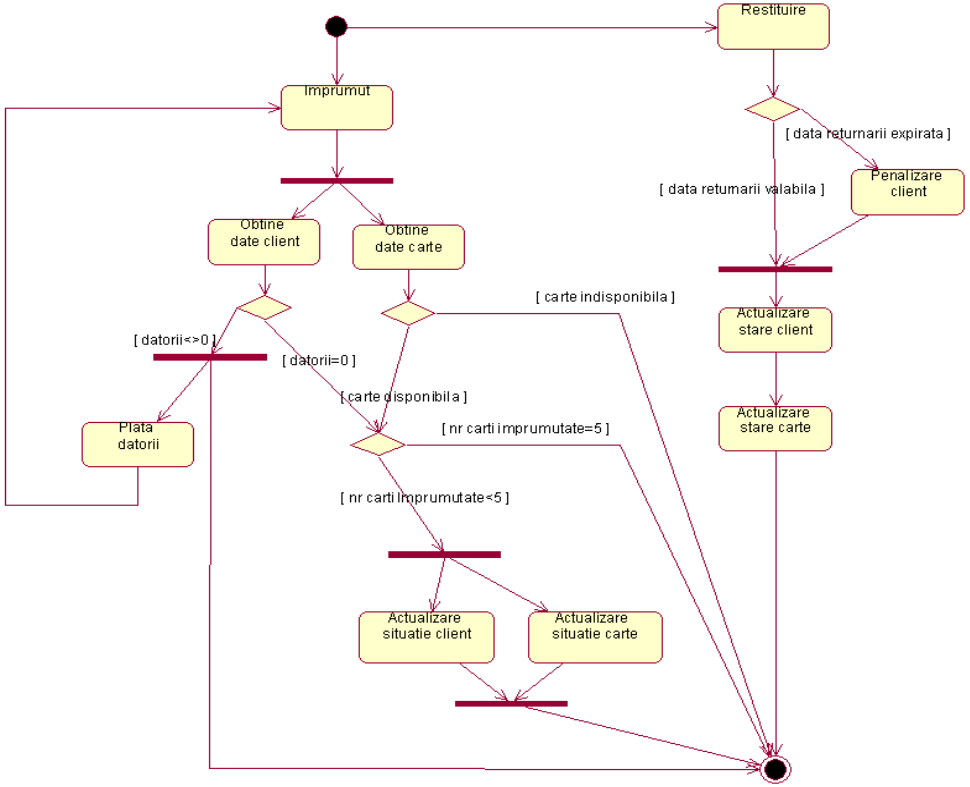


Fig. 4.13. Diagrama de activitate pentru sistemul “Biblioteca”

Pentru sistemul Agenţia de turism pe care l-am mai exemplificat anterior, diagrama de activitate este prezentată în figura 4.14.

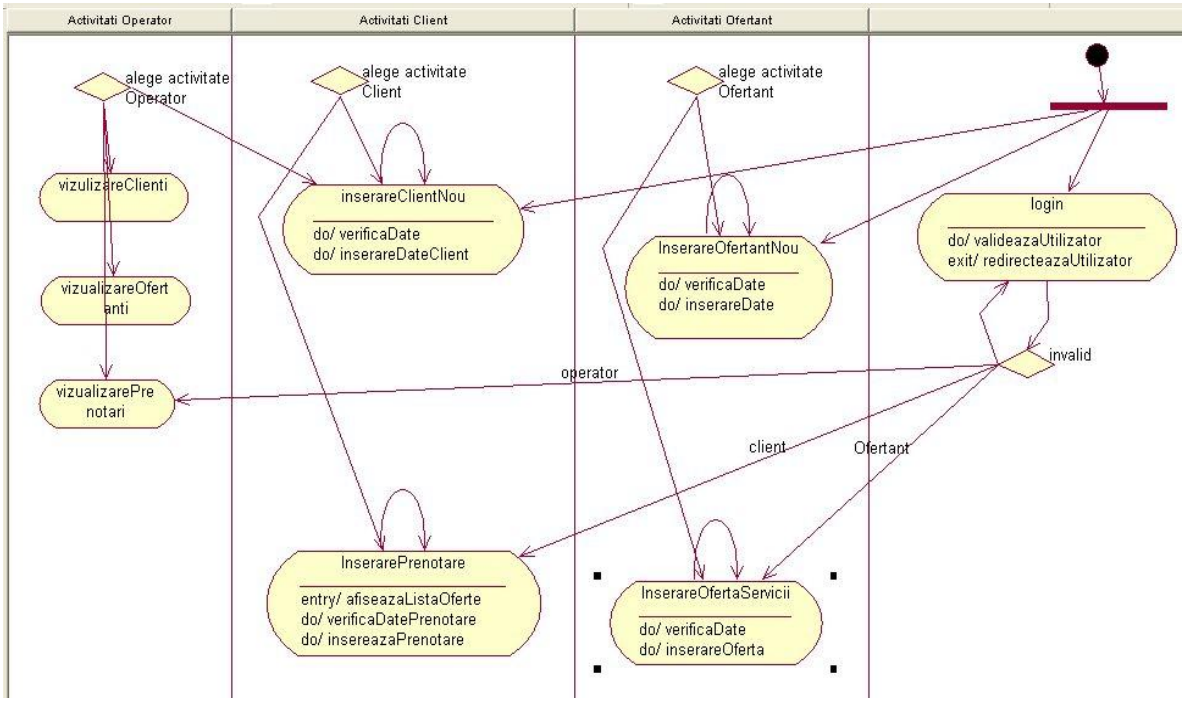


Fig. 4.14. Diagrama de activităţi pentru sistemul Agenţia de turism

Diagrama de activitate reflectă comportamentul mai multor obiecte din cadrul unui caz de utilizare. În figura 4.14 este prezentată de exemplu, activitatea de inserare de client nou şi de logare a acestuia în calitate de client.

Diagrama de activitate poate fi utilizată pentru:

1. Modelarea fluxului de activitate, activităţi aşa cum sunt văzute de actorii din sistem. Aceasta presupune:
   - selectarea obiectelor care au responsabilităţi de nivel înalt pentru fluxul de activitate;
   - identificarea precondiţiilor stărilor iniţiale şi postcondiţiilor stărilor finale;
   - specificarea activităţilor şi acţiunilor începând cu starea iniţială;
   - evidenţierea tranziţiilor care conectează aceste activităţi şi acţiuni;
   - precizarea obiectelor importante implicate în fluxul de activitate, cu evidenţierea schimbării valorilor.
2. Modelarea operaţiilor:
   - colectarea abstracţiilor implicate în operaţii (parametri, atributeale claselor);
   - identificarea precondiţiilor stărilor iniţiale şi postcondiţiilor stărilor finale;
   - specificarea activităţilor şi acţiunilor începând cu starea iniţială;

- folosirea ramificărilor, dacă este necesar;
- folosirea bifurcării şi reunirii pentru specificarea fluxurilor de control paralele.

4.2.3.3. Diagrame de interacţiuni

Diagramele de interacţiuni sunt modele care descriu modul în care grupuri de obiecte colaborează în acelaşi comportament.

De obicei o diagramă de interacţiuni capturează comportamentul unui singur caz de utilizare.

Diagrama prezintă un obiecte şi mesaje care se schimbă între acele obiecte în cazul de utilizare respectiv.

Există două tipuri de diagrame de interacţiune: diagramele secvenţiale şi diagramele de colaborare.

4.2.3.4. Diagramele secvenţiale

Diagramele secvenţiale sunt reprezentări alternative pentru interacţiuni între obiecte.

Ele reprezintă interacţiunile între obiecte din punct de vedere temporal, contextul obiectelor nefiind prezentat în mod explicit (ca în diagramele de colaborare), accentul concentrându-se pe exprimarea interacţiunilor.

Într-o diagramă secvenţială, un obiect este desenat ca un dreptunghi în capătul unei linii verticale întrerupte care reprezintă linia de viaţă a obiectului.

Diagrama de secvenţe, alături de diagrama de colaborare, surprinde colaborările între obiecte în cadrul unui anumit scenariu.

Obiectivul principal al acestei diagrame este acela de a exprima *fluxul* mesajelor între obiecte, în timp secvenţial.

Diagrama de secvenţe arată ordonarea în timp secvenţial a interacţiunilor între obiecte, iar în particular, aceasta arată obiectele care participă la o interacţiune şi succesiunea mesajelor care sunt schimbate.

Modelarea fluxului de control prin ordonarea în timp a mesajelor presupune:

- Stabilirea contextului interacţiunii (sistem, subsistem, operaţie, clasă, un scenariu al unui caz de utilizare sau colaborare);
- Identificarea obiectelor care joacă rol în acţiune;
- Stabilirea pentru fiecare obiect a duratei de viaţă în timpul interacţiunii. Pentru obiectele create şi distruse în timpul interacţiunii trebuie să se indice explicit, prin mesaj, acest lucru;

- Pentru fiecare mesaj începând cu primul (care iniţiază acţiunea) şi continuând cu celelalte în ordinea succesiunii, se prezintă proprietăţile (parametrii);
- Timpul şi spaţiul cerut pentru fiecare mesaj;
- Precondiţii sau postcondiţii pentru fiecare mesaj.

Pentru un flux complet de control se pot utiliza mai multe diagrame.

Diagrama secvenţială are două dimensiuni: una **verticală** care reprezintă timpul, şi una **orizontală** care reprezintă diferite obiecte.

Firele verticale întrerupte reprezintă durata de viaţă a unui obiect.

Mesajele indicate pe săgeţile ce intră într-un anumit fir nu sunt altceva decât metode ale clasei obiectului respectiv, care au fost apelate în cadrul obiectului cu rol de control.

Aşa cum am precizat deja în interiorul unei diagrame secvenţiale, obiectele sunt plasate la începutul diagramei. Dedesubtul fiecărui obiect se află o linie întreruptă, care este numită *linia de viaţă a obiectului*, şi care reprezintă durata de viaţă a obiectului în timpul interacţiunii.

Fiecare mesaj este reprezentat de o săgeată plasată între liniile de viaţă ale celor două obiecte care interacţionează.

Ordinea în care aceste mesaje sunt transmise este de la începutul către sfârşitul diagramei.

Fiecare mesaj are o etichetă cu numele mesajului; de asemenea se pot include argumente şi unele informaţii de control şi se pot folosi auto-delegaţiile.

Auto-delegaţia este un mesaj pe care un obiect şi-l transmite singur, reprezentat de o buclă întoarsă către linia de viaţă a obiectului.

Un element nou care apare în diagrama secvenţială este ***activarea***.

Activarea se petrece atunci când o metodă este activată, deoarece ea poate să fie în execuţie sau în aşteptare.

Un alt element care apare este *mesajul asincron*. Acest mesaj asincron poate executa unul din următorii paşi:

- creează un nou fir;
- creează un obiect nou;
- comunică cu un fir care este deja în execuţie.

În fig. 4.15 este prezentată diagrama secvenţială pentru acelaşi sistem exemplificat şi în diagrama cazurilor de utilizare şi destinat unei agenţii de turism.

Diagrama a fost realizată în mediul Visual Paradigm.

Dreptunghiurile verticale situate pe liniile de viaţă ale obiectelor reprezintă activarea metodelor.

Simbolurile X prezente la sfârşitul liniilor de viaţă indică ştergerea obiectului.

Săgeţile întoarse reprezintă auto-delegaţia.

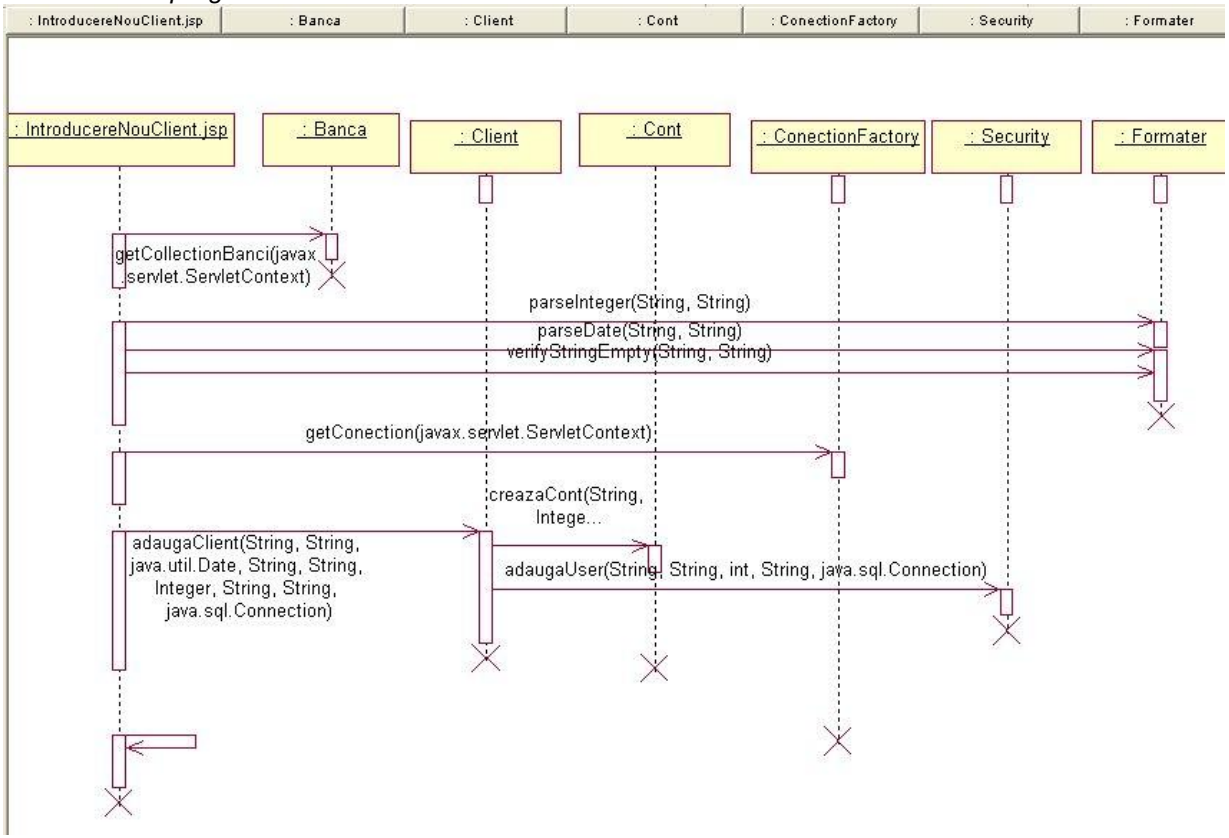


Fig. 4.15. Diagrama secvenţială pentru o agenţie de turism

În concluzie *diagrama secvenţială* trebuie să analizeze detaliat o anumită secvenţă liniară a fluxului de control din cadrul unui caz de utilizare, urmărind un anumit fir de activităţi din diagrama de activităţi, după ce clasele au fost modelate în detaliu .

4.2.3.5. Diagrame de colaborare

Diagrama de colaborare este un tip de diagramă de interacţiune, înrudită cu diagrama secvenţială, cu diferenţa că, în acest caz, accentul cade pe interacţiunea (comunicarea prin schimb de mesaje) între diferitele obiecte implicate într-un caz de utilizare şi nu pe succesiunea în timp a mesajelor .

Secvenţialitatea acestora poate fi totuşi modelată prin numerotare (nu prin dispunerea de-a lungul axei care simboliza durata de viata a unui obiect) .

Fiecare diagramă de colaborare realizează o vedere de ansamblu a legăturilor sau a relaţiilor structurale ce se stabilesc între obiectele şi entităţile obiectelor din modelul curent.

Se pot crea una sau mai multe diagrame de colaborare pentru fiecare pachet logic din model.

Elementele de bază ale acestui tip de diagramă sunt obiectele (instanţe ale claselor), legăturile (instanţe ale asocierilor definite între clase în diagrama claselor), şi mesajele care pot fi asociate bidirecţional legăturilor.

Un obiect are: stare, comportament şi identitate.

Fiecare obiect din diagramă indică o instanţă a clasei.

Simbolul de obiect este similar cu cel al clasei exceptând faptul că numele este subliniat.

Dacă se utilizează acelaşi nume pentru mai multe obiecte utilizate în aceeaşi diagramă, ele se presupun a reprezenta acelaşi obiect; pe de alta parte, fiecare simbol de obiect reprezintă în mod distinct un obiect.

Dacă există mai multe instanţe de obiecte ale aceleaşi clase, se poate modifica simbolul de obiect de exemplu, cu un click pe opţiunea Multiple Instances din Object Specification, al meniului mediului UML.

Mesajele dintre obiecte reprezintă comunicarea dintre acestea şi indică acţiunea în desfăşurare. Mesajul se scrie orizontal pe o săgeată ce face legătura dintre două obiecte.

Un mesaj se poate reprezenta în trei moduri: mesajul singur, mesajul însoţit de numărul secvenţei sau mesajul cu numărul secvenţei şi o etichetă.

Să luăm ca exemplu un sistem de gestionare a unei biblioteci departamentale şi să întocmim diagrama de colaborare pentru : scenariul de împrumut a unei cărţi şi scenariul de restituire a cărţii. Fig. 4.16. reprezintă diagrama de colaborare pentru scenariul de împrumut al unei cărţi, iar figura 4.17 prezintă diagrama de colaborare pentru scenariul de restituire a cărţii.

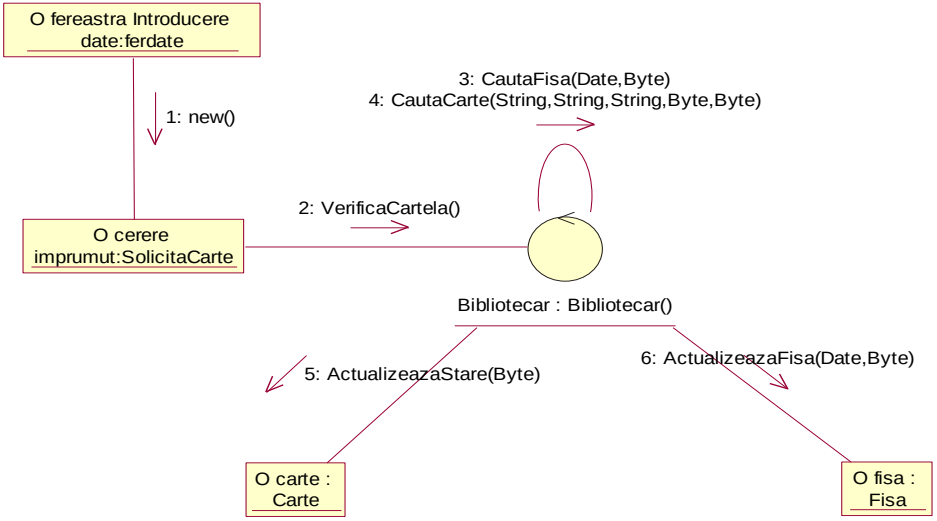


Fig. 4.16. Diagrama de colaborare pentru împrumut de carte dintr-o bibliotecă

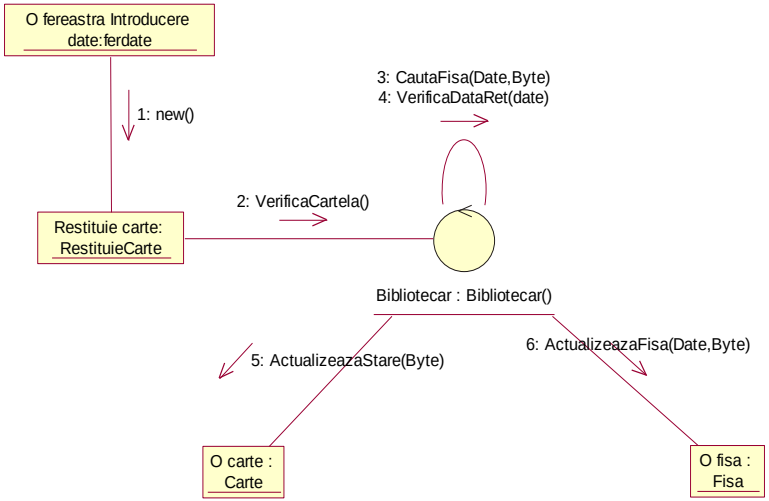


Fig. 4.17. Diagrama de colaborare pentru scenariu de restituire a unei cărţi

Pentru o mai bună înţelegere şi eventual o comparaţie între cele două tipuri de diagrame de interacţiuni (diagrama secvenţială şi diagrama de colaborare) figura 4.18. reprezintă diagrama de colaborare pentru sistemul destinat agenţiei de turism discutat anterior.

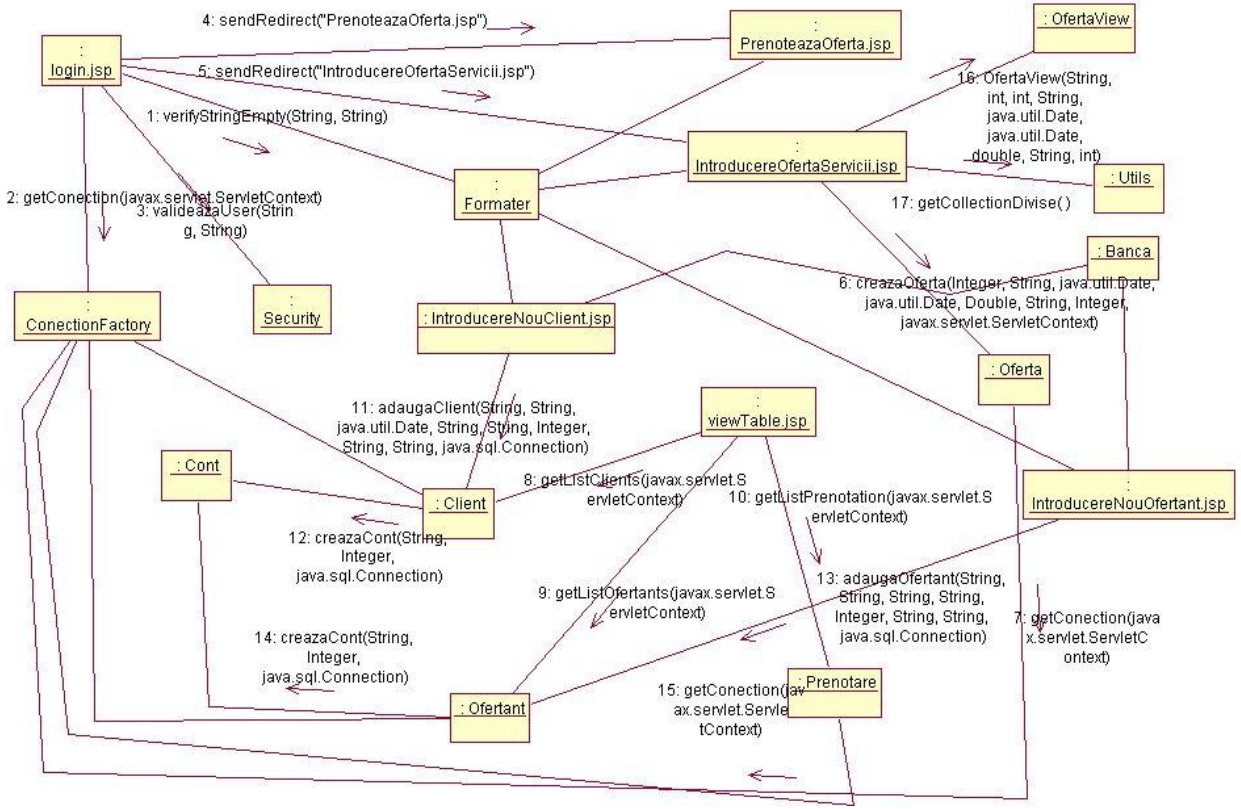


Fig. 4.18. Diagrama de colaborare pentru agenţia de turism

4.2.4. Diagrame de implementare

Diagramele de implementare, aşa cum le arată şi numele, relevă aspecte ale implementării, incluzând cod sursă şi execuţia.

Există două tipuri de astfel de diagrame şi anume: diagrame de componente şi diagrame de aplicaţie.

4.2.4.1. Diagrama de componente

Diagrama de componente este utilizată în modelarea aspectelor fizice ale sistemelor prezentând modul de organizare şi relaţiile de dependenţă între componente şi obiecte.

O astfel de diagramă are în componenţă următoarele elemente UML: componente, obiecte, interfeţe şi relaţii de dependenţă.

Componenta se va utiliza pentru a reprezenta software-ul utilizat de sistem (cod sursă, cod binar sau executabil) sau alte documente existente în sistem.

O instanţă a unei componente reprezintă o implementare run-time şi poate fi utilizată pentru a arăta implementarea unit-urilor care au identitate în momentul execuţiei aplicaţiei.

Componentele unui sistem, incluzând programe, DLL, etc., pot fi amplasate în noduri.

Pentru a figura dependenţele dintre diferite componente se utilizează o linie întreruptă de la o componentă la alta sau, de la o componentă la interfaţa altei componente.

Alte elemente care pot fi incluse sunt: note, legături, note ataşate.

Diagrama se utilizează în unul dintre următoarele scopuri:

- Modelarea codului sursă;
- Modelarea codurilor executabile;
- Modelarea bazelor de date fizice;
- Modelarea sistemelor adaptabile.

Fig. 4.19. prezintă o diagramă cu trei componente: *Clienţi*, care este o bază de date, *Comenzi*- bază de date, *Evidenţă comenzi*- aplicaţie. De asemenea în diagramă există şi un obiect, *Popescu Ion*, care este instanţă a clasei *Client*. Relaţiile dintre aceste elemente sunt relaţii de dependenţă.

Fig.4.20. prezintă o diagramă cu patru componente: *Comenzi*, *Terţi* şi *Produse*, care sunt fişiere şi *Evidenţă terţi*, care este aplicaţie.

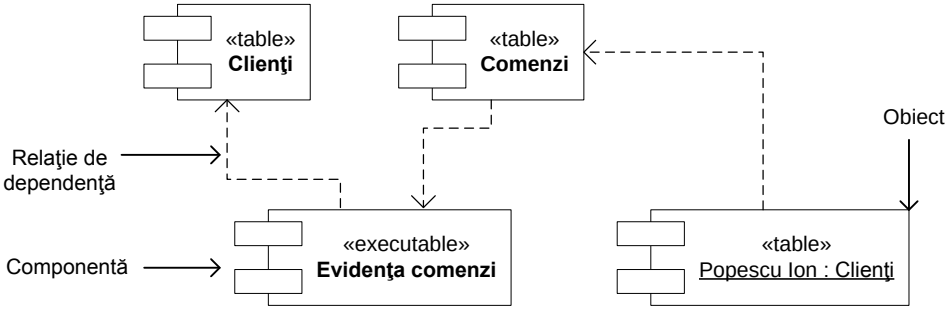


Fig. 4.19. Diagrama cu trei componente

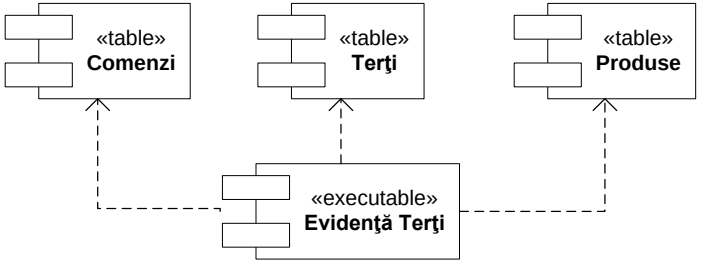


Fig. 4.20. Diagrama cu patru componente

4.2.4.2. Diagrama de aplicaţie (deployment diagram)

Diagramele de aplicaţie sau de desfăşurare arată structura fizică a sistemului hardware şi software, mai precis configurarea elementelor de procese run-time, a componentelor software, şi a proceselor şi obiectelor.

Au în componenţă următoarele elemente UML: noduri, componente, obiecte, relaţii de dependenţă, relaţii de asociere (comunicaţie), precum şi elemente ajutătoare: note, legături, etc.

Componentele au acelaşi rol şi funcţii ca în diagrama componentelor.

Nodurile sunt obiecte fizice care există în timpul execuţiei aplicaţiei şi reprezintă de obicei resurse de prelucrare.

Ele includ componente ale calculatoarelor, resurse umane sau resurse de procesare mecanică.

Diagramele de aplicaţie arată configuraţia elementelor de procesare în timpule execuţiei aplicaţiei, precum şi componente software, procese şi obiecte.

Ele sunt utilizate pentru a modela viziunea asupra desfăşurării statice a sistemului.

Componentele care nu există ca entităţi la execuţie (de exemplu un program care a fost compilat) nu apar în aceste diagrame, ci numai în diagrama componentelor.

O diagramă este reprezentată ca un graf în care nodurile sunt conectate prin relaţii de comunicare.

Nodurile pot conţine şi instanţe ale componentelor.

Acest lucru indică faptul că acele componente există sau rulează în acele noduri.

La rândul lor componentele pot conţine obiecte.

Componentele sunt conectate cu alte componente prin intermediul relaţiilor de dependenţă (linii întrerupte în diagramă), sau prin interfeţe.

Aceste interfeţe indică faptul că o componentă utilizează serviciile unei alte componente.

Componentele pot să migreze de la un nod la altul, iar acest lucru se reprezintă în diagramă cu ajutorul stereotipului << becomes>> pentru relaţii de dependenţă.

O diagramă de aplicaţie, pentru o aplicaţia cu clienţi şi furnizori este prezentată în fig. 4.21.

Diagramele de aplicaţie se folosesc în următoarele cazuri:

- Modelarea sistemelor cu software implantat hardware. Diagramele se utilizează pentru a modela dispozitivele şi procesele care compun sistemul.

- Modelarea sistemelor client-server. Un astfel de sistem este o arhitectură focalizată pe realizarea unei separări nete între interfaţa sistemului cu utilizatorul (de la client) şi datele permanente ale sistemului (de pe server).
- Modelarea sistemelor complet distribuite.

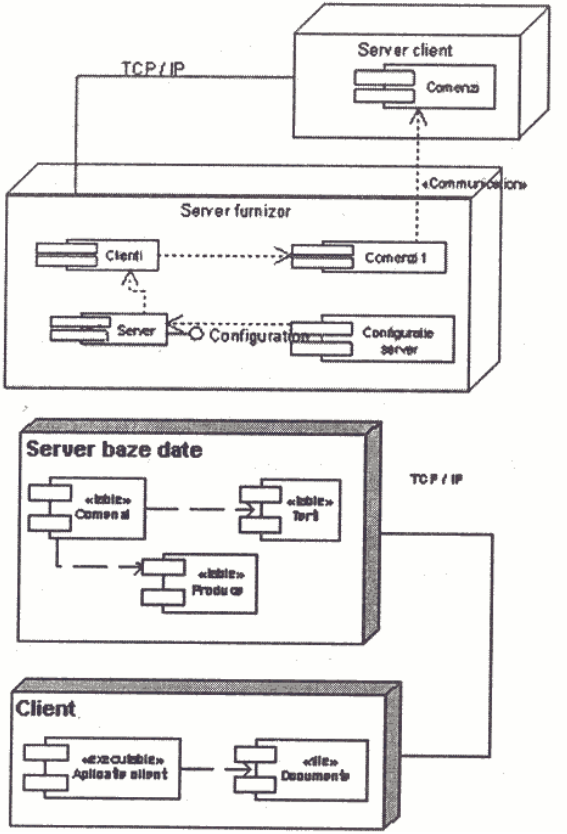


Fig. 4.21. Diagrama de aplicaţie