Modelarea Sistemelor Informatice

UML

UTM 2022

Cuprins

1. Modelarea SI

1.1. Introducere

Obiectivul este familiarizarea cu un număr de modele care descriu acelaşi sistem din diverse puncte de vedere. Aceste modele se dezvoltă în fazele de extragere şi de analiză a cerințelor din cadrul procesului de dezvoltare software.

După parcurgerea acestui capitol vor fi fost înțelese conceptele de modele statice, comportamentale, de date şi de obiecte impreuna cu noțiunile de bază cuprinse în modelele UML (Unified Modeling Language).

Cerinţele utilizatorilor ar trebui să fie scrise în limbaj natural, deoarece acestea trebuie să fie înţelese de către oameni care nu sunt experţi tehnici. Cu toate acestea, cerinţe de sistem mai detaliate pot fi exprimate într-un mod mai tehnic. Tehnica cea mai utilizată în acest moment este de a documenta specificaţiile sistemului sub forma unui set de modele. Aceste modele sunt reprezentări grafice care descriu procesele de afaceri, problema care trebuie rezolvată şi ce sistem urmează să fie dezvoltat. Din cauza reprezentărilor grafice, modelele sunt mai uşor de înţeles decât descrierile detaliate în limbaj natural. Ele sunt, de asemenea, o punte importantă între etapa de analiză şi cea de implementare.

Puteţi folosi modele create în cadrul procesului de analiză pentru a dezvolta o înţelegere a: sistemului existent, a sistemului care urmează să fie înlocuit sau îmbunătăţit sau pentru a specifica un nou sistem care este necesar.

Puteţi dezvolta modele diferite pentru a reprezenta sistemul din diferite perspective. De exemplu. se pot construi modele care să arate:

1. Perspectiva externă, în care se **modelează contextul** sau mediul sistemului;
2. O **perspectivă comportamentală**, în care comportamentul sistemului este modelat;
3. **Perspectiva structurală**, în cazul în care arhitectura sistemului sau structura datelor prelucrate de sistem este modelată.

Cea mai importantă calitate a unui model este faptul că extrage esențialul şi lasă în afară detaliile. Modelul este o abstractizare a sistemului analizat, mai degrabă decât o alternativă de reprezentare a acestuia. În mod ideal, o reprezentare a unui sistem ar trebui să menţină toate informaţiile despre el. O abstractizare, în mod deliberat, simplifică şi preia caracteristicile cele mai importante. De exemplu, în cazul în care o carte este prezentată într-un articol de ziar, prezentarea ar fi o rezumat al punctelor cheie. În cazul în care cartea ar fi tradusă din limba engleză în limba italiană, aceasta ar fi o reprezentare alternativă.

Diferite tipuri de modele de sisteme se bazează pe abordări diferite ale abstractizării. Un model al fluxurilor de date (de exemplu) se concentrează asupra fluxului de date şi al transformărilor pe care le suferă aceste date. Se lasă detalii deoparte detaliile cu privire la structura datelor. Pe de altă parte, un model ERD (al entităţilor de date şi al relaţiilor dintre ele), documentează structura datelor din sistem, mai degrabă decât funcţionalitatea acestuia.

Exemple de tipuri de modele de sisteme care pot fi create în timpul analizei cerințelor:

1. Un *model al fluxurilor de date*: arată modul în care datele sunt prelucrate în diferite etape de către sistem.
2. Un *model de compoziţie* (sau model de agregare) arată modul în care entităţile din sistem sunt compuse din alte entităţi.
3. Un *model de arhitectură*: arată principalele sub-sisteme care compun sistemul.
4. Un *model de clasificare*: arată caracteristicile comune ale entităţilor.
5. Un modelul stimul-răspuns (sau diagrame de tranziţie a stărilor): arată modul în care sistemul reacţionează la evenimentele interne şi externe.

1.2. Modelele de context

Într-un stadiu incipient al etapei de extragere a cerinţelor trebuie luată o decizie cu privire la limitele sistemului. Acest lucru implică lucrul cu părţile interesate de sistem pentru a distinge ceea ce este sistemul şi ceea ce este mediul sistemului. În unele cazuri, graniţa dintre un sistem şi mediul său este relativ clară. De exemplu, în cazul în care un sistem automatizat înlocuieşte unul pre-existent, mediul noului sistem este de obicei acelaşi ca cel al sistemului existent. În alte cazuri, există o mai mare flexibilitate, şi decizia nu mai este aşa simplă. De exemplu, să presupunem că este în curs de dezvoltare specificaţia pentru sistemul de bibliotecă (un sistem care trebuie sa ofere versiuni electronice ale unor cărți protejate de drepturi de autor) din care utilizatorii pot imprima copii personale ale materialului. În elaborarea specificaţiei pentru acest sistem, trebuie decis dacă alte baze de date cu cărți, cum ar fi cataloagele bibliotecii universității, sunt incluse în sistem sau sunt în afara limitelor lui. Dacă sunt, atunci accesul la sistem ar trebui să se facă prin interfaţa de căutare a aplicației bibliotecii universității. Dacă nu sunt, atunci utilizatorii trebuie să treacă de la un sistem la altul prin log-in în ambele aplicații.

Odată ce deciziile cu privire la limitele sistemului au fost făcute, o parte din activitatea de analiză este concentrată pe definiţia acestui context, precum şi dependenţele dintre sistem şi mediului său. În mod normal, producerea un model arhitectural simplu este primul pas în această activitate.

Figura urmatoare reprezinta un model arhitectural, care ilustrează structura de informaţii a unui sistem care include o bancă şi un ATM. Se poate vedea că fiecare ATM este conectat, printre altele,

la o bază de date cu conturile clienților, un sistem contabil al sucursalei, un sistem de securitate şi de un sistem de întreţinere pentru ATM-uri. Sistemul este, de asemenea, conectat la o bază de date care monitorizează modul de funcționare al reţelei de ATM-uri (counter system) care oferă servicii, cum ar fi de backup şi de imprimare. Acestea, prin urmare,nu trebuie să fie incluse în sistemul ATM-ului în sine.

Modele arhitecturale descriu mediul sistemului. Cu toate acestea, nu arată relaţiile dintre alte sisteme din mediul înconjurător şi sistemul care este specificat. Sistemele externe ar putea produce date sau pot consuma date din sistem, ar putea partaja date cu sistemul, ar putea fi conectate direct printr-o reţea sau sa nu fie conectate de loc. Acestea ar putea fi fizic amplasate în zone separate.

Toate aceste relaţii ar putea afecta cerinţele sistemului proiectat şi trebuie să fie luate în considerare. Prin urmare, modelele arhitecturale simple sunt în mod normal, completate de alte modele, cum ar fi modele de proces, care arată activităţile care sunt sprijinite de sistem sau modele ale fluxurilor de date care arată datele care sunt transferate între sistem şi alte sisteme din mediul său.

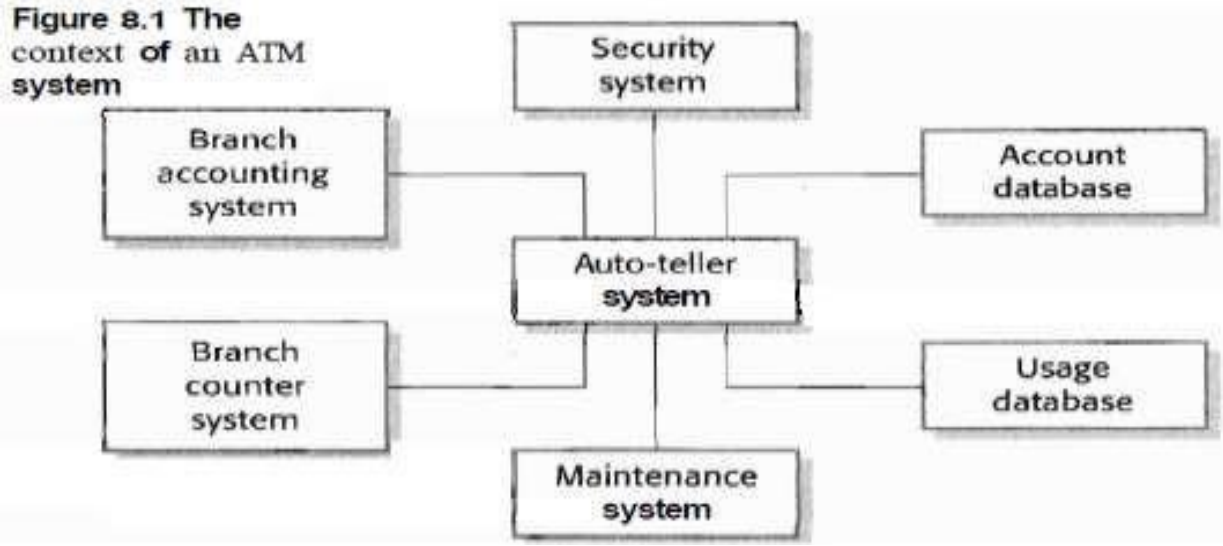
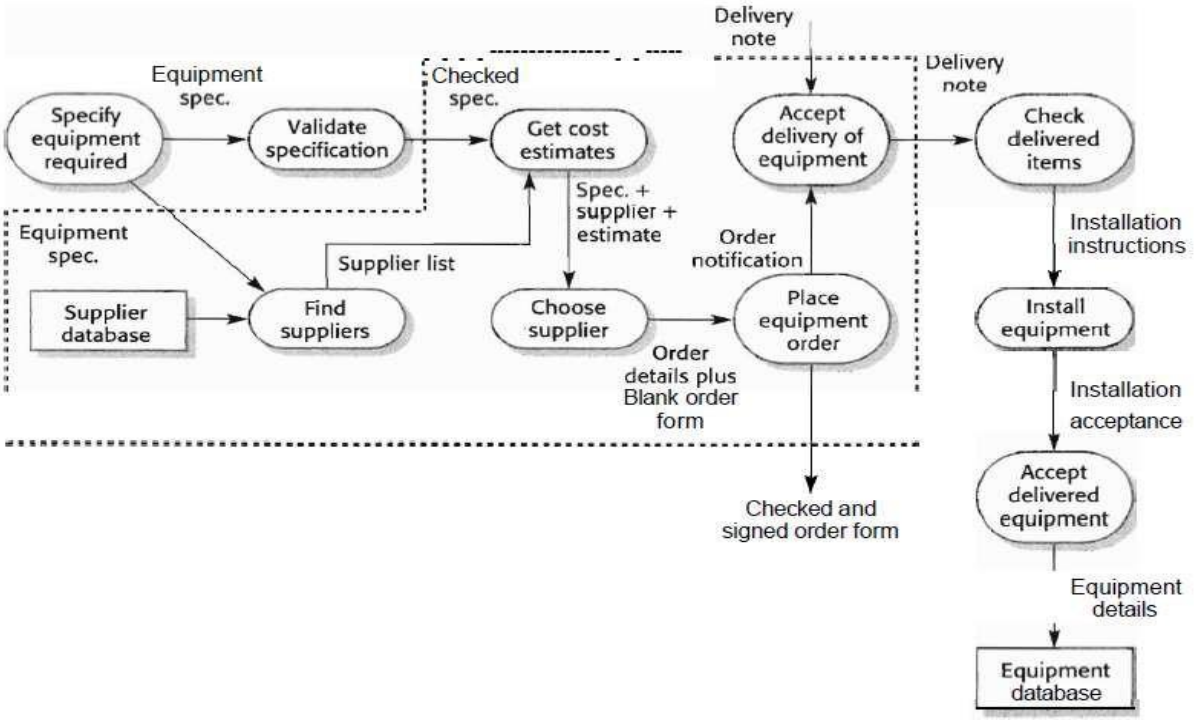


Figure 8.1 The context of an ATM system

Figura următoare ilustrează un model de proces pentru achiziţionarea de echipament într-o organizaţie. Acest lucru presupune specificarea echipamentele necesare, găsirea şi alegerea furnizorilor, comanda echipamente, luând în livrarea de echipamente şi testarea după livrare. La specificarea implicării sistemului în acest proces, va trebui decis care dintre aceste activităţi vor fi sprijinite de fapt şi ce alte activităţi sunt în afara limitei a sistemului. De exemplu, linia punctată demarcă activităţile care sunt în cadrul limitelor sistemului.



1.3. Modelele comportamentale

Modelele comportamentale sunt utilizate pentru a descrie comportamentul general al sistemului. Vom prezenta, pe scurt, două tipuri de modele de comportament: modelele fluxurilor de date (modelează prelucrare datelor în sistem), şi modele stare (modelează modul în care sistemul reacţionează la evenimente). Aceste modele pot fi utilizate separat sau împreună, în funcţie de tipul de sistem care este în curs de dezvoltare.

Cele mai multe sisteme de afaceri sunt bazate, în primul rând, pe date. Ele sunt controlate de datele de intrare ale sistemului, cu relativ puţine procesări ale evenimentelor externe. Un model de flux de date poate fi suficient pentru a reprezenta comportamentul acestor sisteme. Prin contrast, sistemele în timp real sunt adesea determinate de un eveniment extern iar prelucrarea datelor are un caracter minimal. Un model de stare este modul cel mai eficient pentru a reprezenta comportamentul lor.

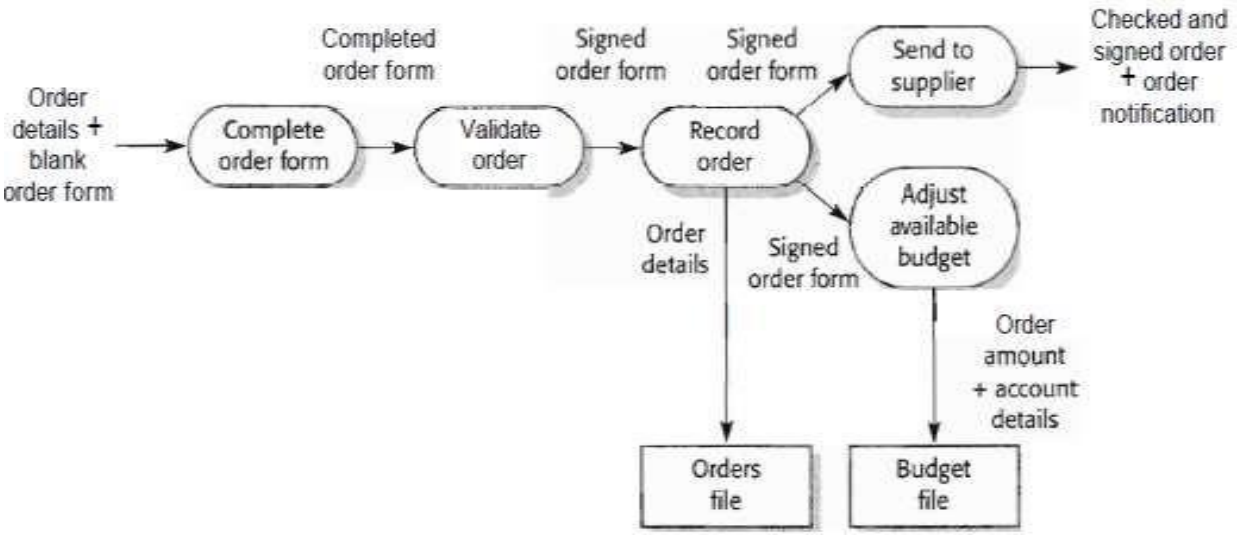
1.4. Modelele fluxurilor de date

Modelele fluxurilor de date sunt o modalitate intuitivă de a arăta modul în care datele sunt prelucrate de către un sistem. La nivel de analiză, acestea ar trebui să fie folosite pentru a modela modul în care datele sunt procesate în sistemul existent. Notaţia utilizată în aceste modele

reprezintă prelucrări de date (dreptunghiuri rotunjite), baze de date (dreptunghiuri) şi mişcări de date între prelucrări (săgeţi etichetate).

Modelele fluxurilor de date sunt folosite pentru a arăta modul în care datele „curg" printr-o secvenţă de paşi de prelucrare. De exemplu, o etapă de prelucrare ar putea fi filtrreaa înregistrărilor dublură dintr-o bază de date cu clienții firmei. Datele se transformă la fiecare pas înainte de a trece pe la etapa următoare. Aceste etape de prelucrare sau de transformări reprezintă procese software sau funcţii, atunci când modelele fluxurilor de date sunt utilizate pentru a documenta un proiect software. Cu toate acestea, într-un model de analiză, atât oamenii cât şi calculatoarele pot efectua prelucrări.

Un model al fluxurilor de date care arată paşii implicaţi în procesarea unui ordin de achiziție de bunuri (cum ar fi un calculator), într-o organizaţie, este ilustrat în figura următoare. Acest model descrie prelucrarea datelor, în scopul realizării activității Place Equipment Order din figura de mai sus. Modelul arată cum pentru se mișcă comanda de achiziție de la o activitate a procesului general (din figura de mai sus) la o altă activitate. Ea arată, de asemenea, structurile de date (tabelele Orders si Budget) care sunt implicate în acest proces.



În principiu, dezvoltarea modelelor fluxurilor date ar trebui să fie un proces "top-down". În acest exemplu, acest lucru ar presupune că ar trebui să înceapă prin analizarea procesul general de achiziţie. Puteţi trece apoi la analiza sub-proceselor cum ar fi comanda. În practică, analiza nu este niciodată aşa. Veţi afla lucruri despre mai multe niveluri ale procesului, în acelaşi timp. Uneori poate fi mai convenabilă construirea unui nivel de modelele detaliat şi apoi folosită abstractizarea pentru a crea un model mai general.

Modelele fluxurilor de date arată o perspectivă funcţională în care fiecare transformare reprezintă o singură funcţie sau proces. Ele sunt utile în special în timpul analizei cerinţelor deoarece pot fi utilizate pentru a modela prelucrare datelor de la început până la sfârşit. Adică, aceste diagrame arată întreaga secvenţă de procesări care au loc astfel încât o intrare a sistemului să fie prelucrată într-o ieşire corespunzătoare (care este răspunsul sistemului la un eveniment/cerință).

1.5. Modelele de stare

Un model de stare descrie modul în care un sistem răspunde la evenimente interne sau externe. Modelele de stare arată stările sistemului şi evenimentele care cauzează tranziţii de la la o stare la alta. Aceasta nu arată fluxul de date în cadrul sistemului. Acest tip de model este adesea utilizat pentru modelarea sistemelor in timp real, deoarece aceste sisteme sunt de multe ori conduse de stimuli din mediul extern. De exemplu, un sistem de alarma în timp real răspunde la stimuli cum ar fi: senzori de mişcare, senzori de deschidere a uşilor, etc.

Un model de stare a unui sistem presupune că, în orice moment, sistemul este într-una dintre o serie de stări posibile. Atunci când un stimul este primit, aceasta poate declanşa trecerea la o stare la altă stare. De exemplu, un sistem de control al unui robinet se pot deplasa de la starea "Robinet deschis" la o altă stare "Robinet închis", atunci când un operator dă comanda (stimulul) de închidere a alimentării cu apă.

Diagramele de stare vor fi discutate în contextul UML din partea a doua a acestui capitol.

1.6. Modele de date

Cele mai multe sisteme de business utilizează o baza de date de mari dimensiuni. În unele cazuri, această bază de date este independentă de sistemul software. În altele, acesta este creată strict pentru uzul sistemului. În aceste cazuri, o parte importantă a activității de modelare este definirea formei logice a datelor prelucrate de sistem. Acestea sunt uneori numite modelele semantice de date.

Cele mai utilizată tehnică de modelare de date este entitate-relaţie-atribut (ERD), care arată entităţile de date, atributele asociate acestora şi relaţiile dintre aceste entităţi. Această abordare de modelare a fost propus pentru prima oară în mijlocul anilor 1970 şi mai multe variante au fost dezvoltate de atunci, toate având la bază acelaşi concepte (entitate şi relaţie) şi fiind utilizate pentru proiectarea bazelor de date relaționale. Schemele de baze de date derivate din aceste modele sunt în mod natural în forma a treia normală, care este o caracteristică de dorit. Datorită reprezentării explicite a elementelor (entități, atribute şi relații) şi recunoaşterea de sub si super-tipuri, implementarea este foarte facilă.
UML nu include o notaţie specifică pentru aceste modele de baze de date, deoarece se presupune că se foloseşte un proces de dezvoltare orientat-obiect şi, în consecință, modelează datele folosind noțiunea de obiecte şi relaţiile dintre ele. Cu toate acestea, aveţi posibilitatea să reprezentați în

UML un model semantic de date. Puteţi să vă gândiţi că entităţile dintr-un model ERD sunt clase de obiecte simplificate (care nu au operaţiuni), atributele ca atribute ale clasei şi relaţiile ca asocieri între clase. Mai multe detalii despre diagramele de clase în a doua parte a acestui capitol.

2. Modelele obiectelor

O abordare orientată-obiect a întregului proces de dezvoltare software este acum frecvent utilizată, în special pentru dezvoltarea sistemelor de interactive. Acest lucru înseamnă exprimarea cerinţelor de sisteme folosind modele ale obiectelor, realizarea design-ului pe baza obiectelor şi dezvoltarea sistemului într-un limbaj de programare orientat-obiect, cum ar fi Java sau C++. Modelele obiectelor dezvoltate in timpul analizei cerinţelor pot fi utilizate pentru a reprezenta atât datele sistemului cât şi prelucrarea acestora.

Pentru unele clase de sisteme, modelarea sub forma de obiecte este un mod natural de modelare a realității. Această afirmație este adevărată mai ales când sistemul procesează informații despre obiecte tangibile (ce ex. maşini, avioane sau cărți) care au atribute clar identificabile. Entitățile mai abstracte cum ar fi conceptele de bibliotecă sau procesare de text sunt mai dificil de modelat ca obiecte.

O clasă de obiecte este o abstractizare a unui set de obiecte care împart un set comun de atribute (cum sunt instanțele unei entități dintr-o diagramă ERD) şi de servicii sau operații. Obiectele sunt entități executabile care au atributele şi serviciile clasei de obiecte din care fac parte. Obiectele sunt instanțe ale claselor şi mai multe obiecte pot fi create din aceeaşi clasă. În general, în etapa de analiză a cerințelor se caută identificarea claselor şi a relațiilor dintre acestea.

Dacă se abordează faza de determinare şi analiză a cerințelor dintr-o perspectivă orientată-obiect entitățile din lumea reală ar trebui modelate ca şi clase de obiecte. Modelele nu ar trebui să includă detalii ale obiectelor. Modelele ar trebui să includă informații legate de modul în care clasele sunt legate unele de altele, cum obiectele sunt agregate sau cum obiectele interacționează unele cu altele.

Diverse metode de analiză orientată-obiect au fost propuse in anii 90. Acestea aveau multe elemente comune iar trei dintre cele mai importante (promovate de trei figuri proeminente din domeniu: Booch, Rumbaugh şi Jacobsen) au fost integrate sub forma unei metode unificate. Limbajul de modelare unificat (UML – Unified Modeling Language) folosit în cadrul acestei metode unificate a devenit standardul pentru modelarea orientată-obiect.

2.1. Introducere în UML (Unified Modeling Language)

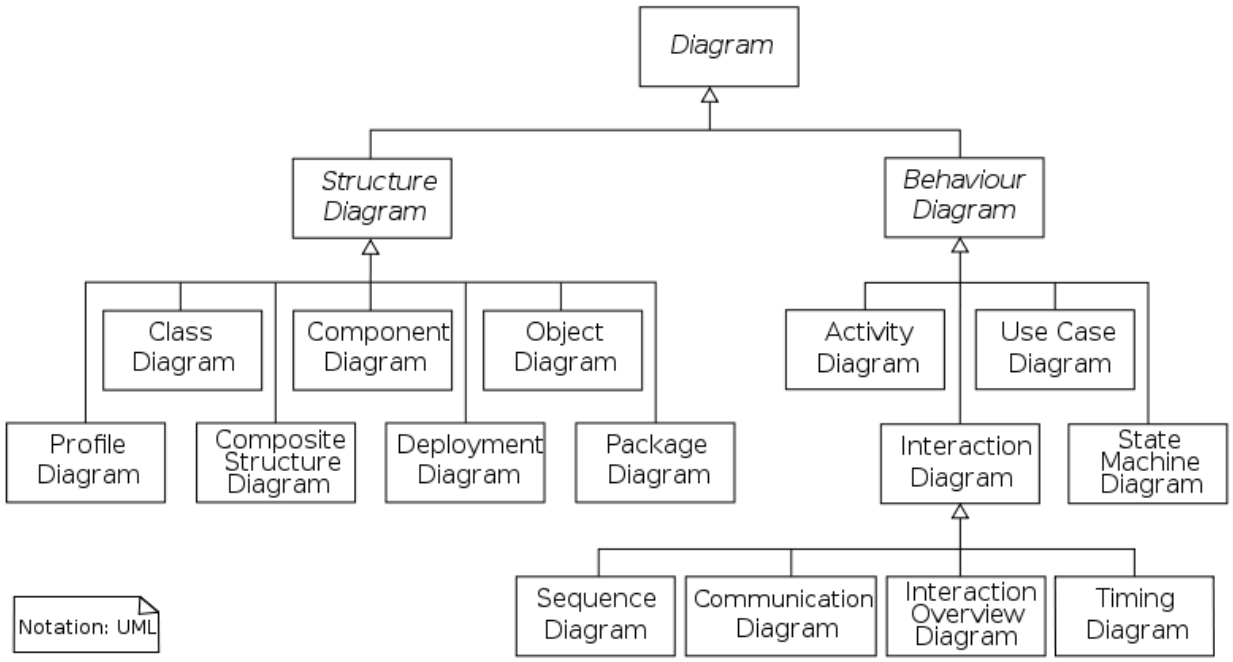
Faza de analiză si proiectare a unui software trebuie să fie realizată înainte de realizarea implementării. Aceste etape au fost ignorate în anumite momente din trecut, dar în prezent orice

dezvoltator recunoaște importanta acestor faze, deoarece s-a dovedit ca de acestea depinde succesul procesului de dezvoltare software. Pentru realizarea comunicării între diversele părți interesate (dezvoltatori, clienți, utilizatori) a rezultatelor analizei şi proiectării se folosesc limbajele de modelare. Cel mai folosit este limbajul de modelare unificat – UML.

În prezent, UML este limbajul universal standard pentru dezvoltatorii software din toata lumea. UML este succesorul propriu-zis al celor mai bune trei limbaje de modelare anterioare orientate pe obiecte (Booch, OMT, şi OOSE). UML se constituie din unirea acestor limbaje de modelare şi, în plus, deține o expresivitate care ajută la rezolvarea problemelor de modelare pe care vechile limbaje nu o aveau.

UML sprijină analiza şi proiectarea sistemelor printr-un limbaj corespunzător pentru: specificarea; vizualizarea: construirea si documentarea artefactelor sistemelor software; şi, de asemenea, pentru modelarea întreprinderii. UML este un limbaj de modelare care oferă o exprimare grafică a structurii şi comportamentului software. Pentru această exprimare grafică se utilizează notațiile UML.

Notațiile UML constituie un element esențial al limbajului pentru realizarea propriu-zisa a modelarii si anume partea preocupată de reprezentarea grafică. Modelarea în acest limbaj se realizează prin combinarea notațiilor UML în cadrul elementelor denumite diagrame. În cadrul UML 2.0 există 15 tipuri de diagrame, dintre care cele mai importante sunt: diagrama cazurilor de utilizare, diagrama de secvența, diagrama de colaborare, diagrama de clase (cea mai utilizata), diagrama de stări, diagrama de componente, diagrama de construcție, diagrama de obiecte şi diagrama de activităţi. Perspectiva generală este prezentată în figura următoare:



Analiza unei aplicaţii implică realizarea mai multor categorii de modele, dintre care cele mai importante sunt:

- Modelul de utilizare. realizează modelarea problemelor şi a soluţiilor acestora în maniera în care le percepe utilizatorul final al aplicaţiei. Diagramă asociată: diagramă de cazuri de utilizare.
- Modelul structural: se realizează pe baza analizei statice a problemei şi descrie proprietăţile statice ale entităţilor care compun domeniul problemei. Diagrame asociate: diagramele de clase, diagramele de obiecte, diagramele de pachete, diagramele de componente, diagramele de implementare, diagramele de structură.
- Modelul comportamental: priveşte descrierea funcționalităților şi a succesiunii în timp a acţiunilor realizate de entităţile domeniului problemei. Diagrame asociate: diagramele de activități, diagramele caturilor de utilizare, diagramele de secvențe, diagramele de stări (state-machine), diagrama de comunicare, diagrama de interacţiune, etc.

2.2. Diagrame UML

O diagramă oferă utilizatorului un mijloc de vizualizare şi de manevrare a elementelor de Modelare. Majoritatea diagramelor se prezintă sub forma unor grafuri, compuse din elemente şi arce.

Diagramele pot arăta o parte sau toate caracteristicile elementelor de modelare, conform nivelului de detaliu util în contextul unei diagrame date. Diagramele pot grupa informații interdependente, pentru a arata, de exemplu caracteristicile moștenite de o clasa. Diagramele esențiale în UML sunt:

- diagrame cazurilor de utilizare, care prezintă funcțiile sistemului din punct de vedere al utilizatorului;
- diagramele claselor, care prezintă structura statică în termeni de clase si asocieri (relații);
- diagrame de colaborare, care sunt reprezentări spațiale ale obiectelor, legăturilor şi interacțiunilor;
- diagrame de secvență, care prezintă temporal obiectele si interacțiunile lor;
- diagrame de componente, care prezintă componentele fizice ale unei aplicații;
- diagrame de stări-tranziții, care prezintă comportamentul unei clase în termeni de stări;
- diagrame de obiecte, care prezintă obiectele şi relațiile lor, fiind niște diagrame de colaborare simplificate, fără reprezentarea mesajelor trimise între obiecte;

- **diagrame de activități**, care reprezintă comportamentul în termeni de acțiuni.

2.3. Modelul de utilizare

Diagramele cazurilor de utilizare (Use-case)

O diagrama a cazurilor de utilizare prezintă o colecție de cazuri de utilizare şi actori si este folosită, în general, pentru a indica sau caracteriza funcționalitățile şi comportamentul întregului sistem interacționând cu unul sau mai mulți actori. Utilizatorii şi orice alt sistem ce poate interacționa cu sistemul sunt actori. În general, actorii reprezintă utilizatorii. Ei ajută la delimitarea sistemului şi oferă o imagine clară a ceea ce se așteaptă a se întâmpla în sistem. Cazurile de utilizare sunt construite pe baza nevoilor pe care le au actorii (utilizatorii). Aceasta asigura faptul ca sistemul va produce ceea ce s-a dorit.

Diagramele cazurilor de utilizare conțin elemente ce pot reprezenta actori, relații de asociere, relații de generalizare, pachete şi cazuri de utilizare. Se poate crea o diagrama a cazurilor de utilizare de nivel înalt, pentru a vizualiza limitele si comportamentul sistemului. Dar, se pot crea una sau mai multe diagrame de nivel jos (de detaliu) pentru a descrie mai bine o parte a sistemului. Un caz de utilizare poate include alte cazuri de utilizare ca o parte a comportamentului său.

Un *actor* poate fi unul din tipurile de utilizatori dar şi orice sistem care poate interacționa cu sistemul. Astfel, un actor reprezintă un rol jucat de o persoana sau o entitate care interacționează cu sistemul. Aceeași persoana fizica poate juca rolul mai multor actori în realitate, aşa cum şi mai multe persoane pot juca același rol, si astfel interacționa cu sistemul ca un singur actor reprezentat în diagramă. Fiecare actor trebuie sa aibă un nume, iar numele acestuia descrie rolul jucat de către actor.

In diagrama cazurilor de utilizare se pot desena *asocieri* de la actor la cazul de utilizare sau *generalizări* între cazuri de utilizare/actori.

- *Asocierea* reprezintă o conexiune semantica intre cazurile de utilizare si actori. Asocierile nu sunt direcționate (sunt reprezentate prin linii care nu au la capete săgeți). Ele sunt relațiile cele mai generale si cele mai slabe din punct de vedere semantic. O asociere poate avea nume si un stereotip care să identifice tipul sau semnificația relației. Relațiile de asociere se pot desena între cazuri de utilizare şi actori.
- *Generalizarea* între două cazuri de utilizare indică faptul că se poate împărtăși comportamentul definit în unul sau mai multe cazuri de utilizare. O generalizare între actori arata ca un actor moștenește structura si comportamentul unui actor sau alor mai mulți

actori. Stereotipul folosit este <<extinde>>. Generalizarea se reprezintă printr-o săgeata ce leagă două elemente (cazuri de utilizare si actori):

Un *caz de utilizare* are un nume, care este caracteristic si nu un nume simplu. Numele unui caz de utilizare începe de obicei cu un verb. Fiecare apariție a unui caz de utilizare în Diagrama Cazurilor de Utilizare indică existenţa un set de informații despre cazul de utilizare. Aceste informații detaliate ale modelului pot fi vizualizate in specificațiile cazului de utilizare.

O notă asociată unei diagrame cuprinde ipotezele şi deciziile aplicate în timpul analizei şi a reprezentării grafice. Notele pot conține orice informație, inclusiv textul planului de dezvoltare software, fragmente de cod sau referințe la alt document. Notele se comporta ca niște etichete. Ele se pot folosi, de altfel, in orice fel de diagrama. Notele sunt doar explicații oferite acolo unde apar in diagrama. Ele nu sunt considerate ca făcând parte din model.

2.4. Modelul structural

Modelul structural descrie structura sistemului (prin diagrame de componente) şi a obiectelor care fac parte dintr-un sistem: identitatea lor, relaţiile cu celelalte obiecte, atributele şi operaţiile lor (numit şi model obiect) prin intermediul diagramelor de clase şi de obiecte.

Câteva dintre conceptele importante folosite în construirea modelului obiect sunt: obiect, clasă, legătură, asociere, agregare şi generalizare.

a. Scopul modelării obiect este de a descrie obiecte. Un obiect se defineşte ca fiind un concept, abstracţie sau lucru cu înţeles şi limite bine definite pentru problema în lucru. Descompunerea unei probleme în obiecte nu este unică şi ea depinde de natura problemei şi de obiectivele care trebuie atinse de cei care realizează modelarea.

b. O clasă descrie un grup de obiecte cu proprietăţi similare (atribute), comportament comun (operaţii) şi relaţii similare cu celelalte obiecte. Gruparea obiectelor în clase realizează o abstractizare a problemei. Abstractizarea permite generalizarea anumitor cazuri specifice la o mulţime de cazuri similare.

c. Un atribut este o trăsătură a obiectelor dintr-o clasă. Fiecare atribut are o anumită valoare pentru fiecare instanţă. Fiecare nume de atribut este unic în cadrul unei clase, dar nu este neapărat unic de-a lungul tuturor claselor.

d. O operaţie este o funcţie sau o transformare care poate fi aplicată unor sau de către obiectele unei clase. Fiecare operaţie are ca argument implicit obiectul ţintă. Aceeaşi operaţie poate fi aplicată în clase diferite şi de aceea poate avea forme diferite pentru fiecare clasă. O metodă este implementarea unei operaţii întro clasă.

e. O legătură este o conexiune fizică sau conceptuală între două sau mai multe instanţe.

f. O asociere descrie un grup de legături cu semantică şi structură comună. Asocierile sunt adesea implementate în limbajele de programare ca pointeri de la un obiect la altul. Asocierile sunt

importante deoarece pot modela foarte bine informaţiile care nu sunt subordonate unei singure clase dar depind de două sau mai multe clase. O noţiune care apare legată de asocieri este multiplicitatea care specifică faptul că zero, una sau mai multe instanţe ale unei clase pot avea legături cu o singură instanţă a clasei asociate.

g. Moştenirea este o abstractizare puternică pentru factorizarea similarităţilor între clase. Generalizarea este o relaţie între o clasă şi una sau mai multe versiuni rafinate ale ei.

Clasa care este rafinată se numeşte superclasă, iar versiunile rafinate ale ei subclase. Atributele şi operaţiile comune unui grup de subclase sunt ataşate superclasei şi sunt partajate de fiecare subclasă, deci spunem că subclasele moştenesc trăsăturile superclasei. Generalizarea este utilă atât la modelare cât şi la implementare. Ea facilitează modelarea prin structurarea claselor şi capturarea succintă a ceea ce este similar şi aceea ce este diferit între clase.

h. Agregarea este o formă puternică a asocierii, în care un obiect agregat este construit din componente, care sunt părţi ale agregatului. Un agregat poate avea mai multe părţi; fiecare relaţie parte-întreg este tratată ca agregare separată. Părţile pot să apară în mai multe agregate. Agregarea este inerent tranzitivă: un agregat are părţi care, la rândul lor, pot avea părţi etc. Agregarea nu este un concept individual ci o formă specială a asocierii. Dacă două obiecte sunt strâns legate printr-o relaţie parte-întreg atunci avem o agregare. În schimb, dacă două obiecte sunt considerate independente, chiar dacă între ele poate exista uneori o relaţie, atunci avem o asociere.

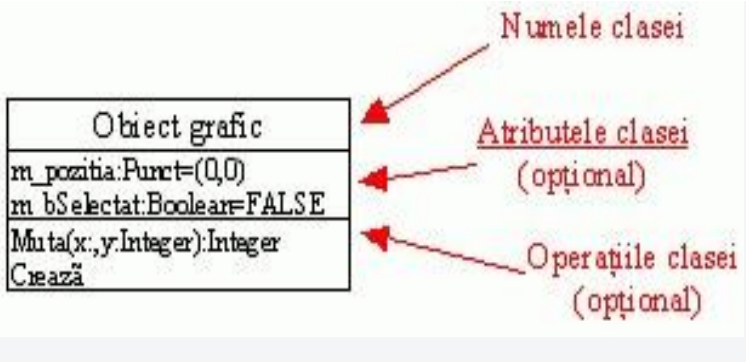
2.4.1. Diagramele de componente

Se folosesc pentru a prezenta componentele unui sistem precum si interfețele (conexiunile) cu alte componente. Componentele pot fi sub-sisteme, module de cod, etc. Aceste diagrame arată modul în care sistemul este „legat”.

2.4.2. Diagramele de clase

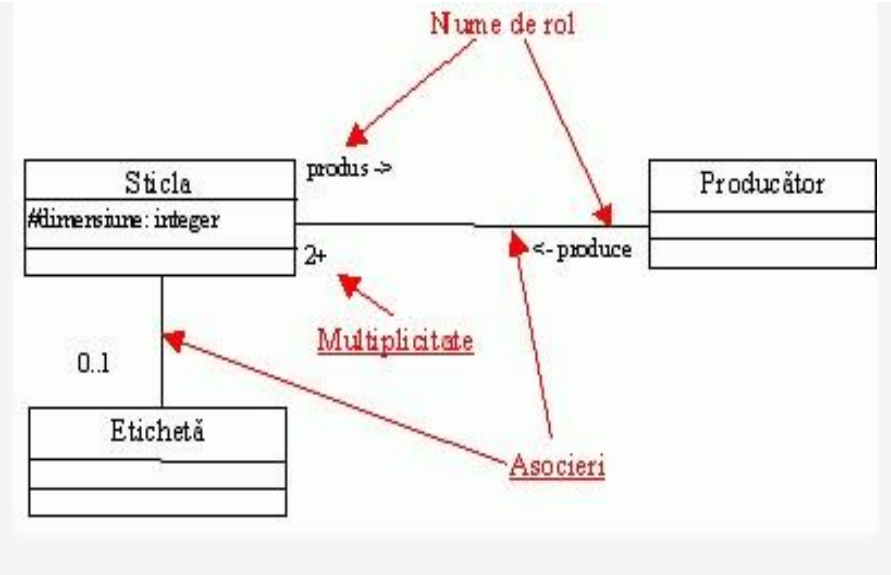
Diagrama folosită în modelarea orientată-obiect se numeşte diagramă de clase şi ea oferă o notaţie grafică pentru reprezentarea claselor şi relaţiilor dintre ele. Diagramele de clase descriu cazuri generale în modelarea sistemului. Clasele sunt reprezentate de dreptunghiuri împărţite în trei compartimente şi care conţin numele clasei (în compartimentul superior), lista de atribute ale clasei (opţional) şi lista de operaţii (opţional) cu argumentele lor şi tipul returnat. Cele trei compartimente vor fi separate între ele prin câte o linie orizontală.

Valorile iniţiale ale atributelor pot fi specificate astfel: nume:tip = expresie.

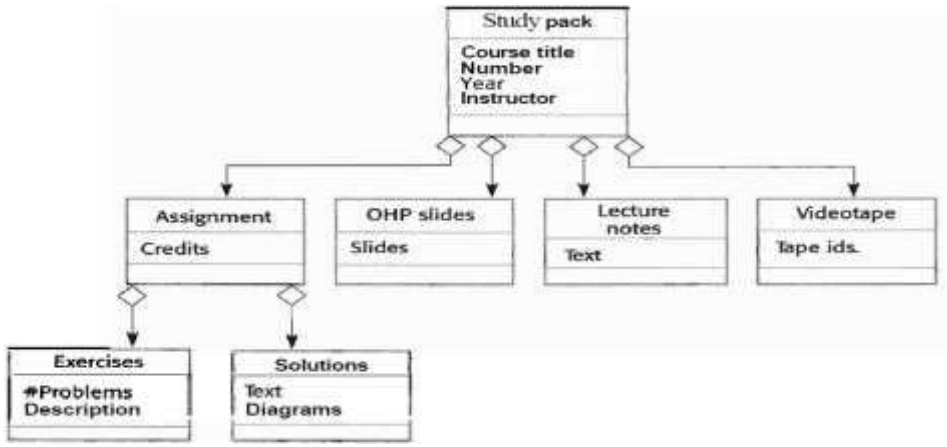


Toate relaţiile din cadrul diagramei de clase sunt reprezentate grafic printr-o succesiune de segmente orizontale sau verticale care leagă o clasă de alta.

Asocierea poate avea sau nu un nume care se va afla tipărit în vecinătatea sa şi va conţine o săgeată care precizează modul de citire al acestuia. O asociere poate avea nume diferite pentru fiecare direcţie. Fiecare capăt al asocierii este un rol şi fiecare rol poate avea un nume (nume de rol) care arată cum este văzută clasa asociată lui de alta. Fiecare rol indică multiplicitatea clasei sale (câte instanţe ale clasei pot fi asociate cu o instanţă a altei clase). Multiplicitatea poate fi 1 (nu se marchează), 0-1 (marcată printr-un cerc alb) 0 sau mai multe (marcată printr-un cerc negru), sau alte intervale de valori întregi care se indică prin expresii de tipul 1+ (mai mult de o instanţă), 3 (exact trei instanţe) sau 2-4 (între 2 şi 4 instanţe inclusiv). Este posibilă asocierea între o clasă şi ea însăşi sau mai multe asocieri la aceeaşi pereche de clase.



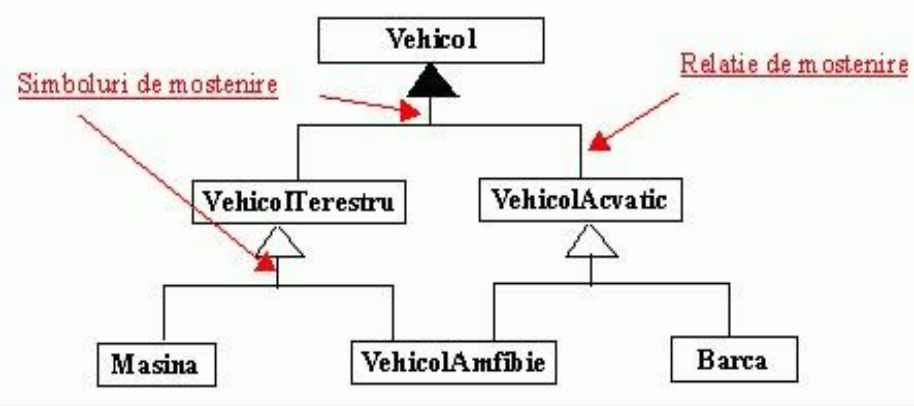
Agregarea este o formă specială de asociere cu înţelesul de relaţie parte-întreg. Ea se indică prin plasarea unui mic romb la capătul corespunzător clasei care semnifică întregul. Multiplicitatea unei agregări precum şi calificarea se reprezintă analog ca la asocieri.



Generalizarea se reprezintă grafic sub forma unui triunghi echilateral cu o linie de la un vârf al acestuia (vârf care trebuie să fie îndreptat spre superclasă) la superclasă şi o linie de la baza triunghiului opusă vârfului la fiecare subclasă.

Generalizare este o relaţie *n*-ară. Din punct de vedere grafic simbolul de generalizare (triunghiul) poate fi considerat ca un nod al diagramei. Deci în reprezentarea unei relaţii de generalizare sunt implicate un nod (triunghiul) şi două arce (unul care uneşte o clasă de vârful triunghiului, şi altul care uneşte o clasă de baza triunghiului).

În general subclasele sunt disjuncte, ceea ce este reprezentat prin culoarea albă a triunghiului; există însă situaţii când acestea nu respectă acestă proprietate, fapt care se reprezintă printr-un triunghi negru.

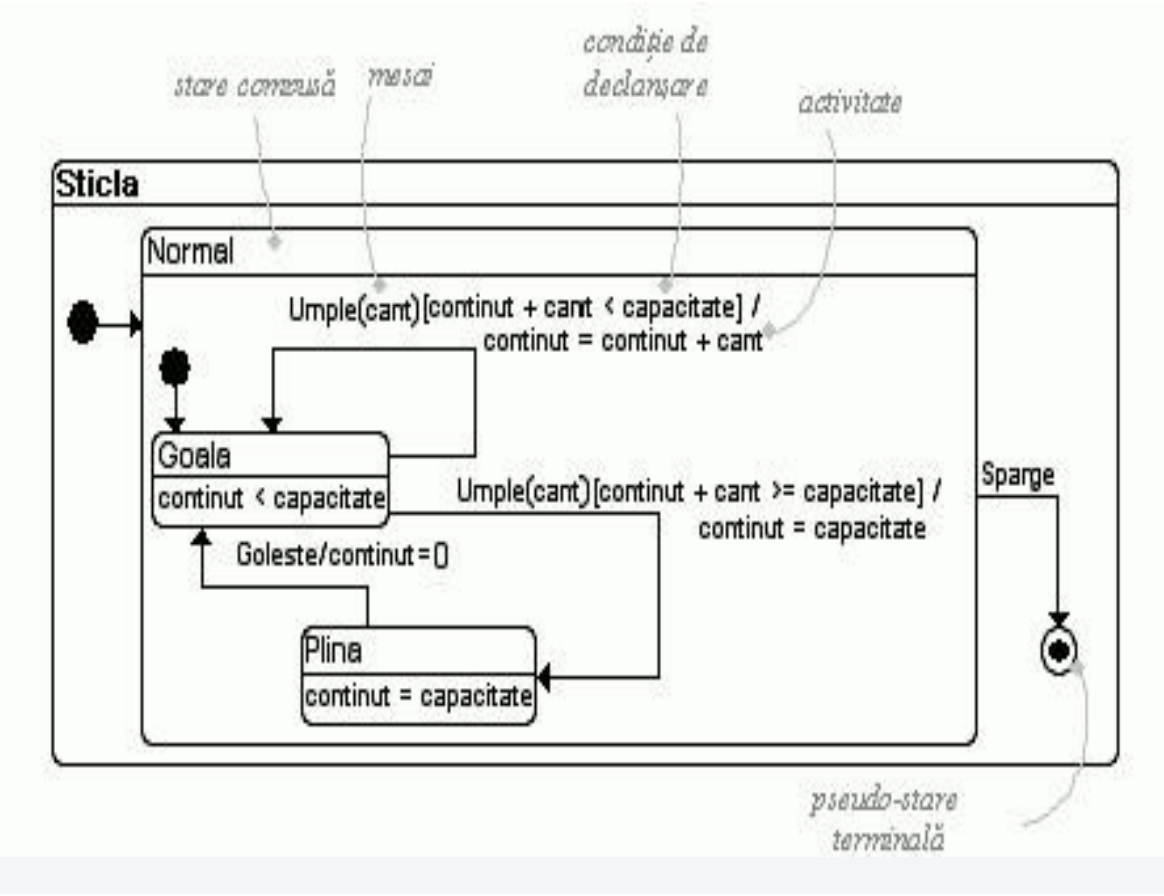


2.4.3. Diagramele de stări

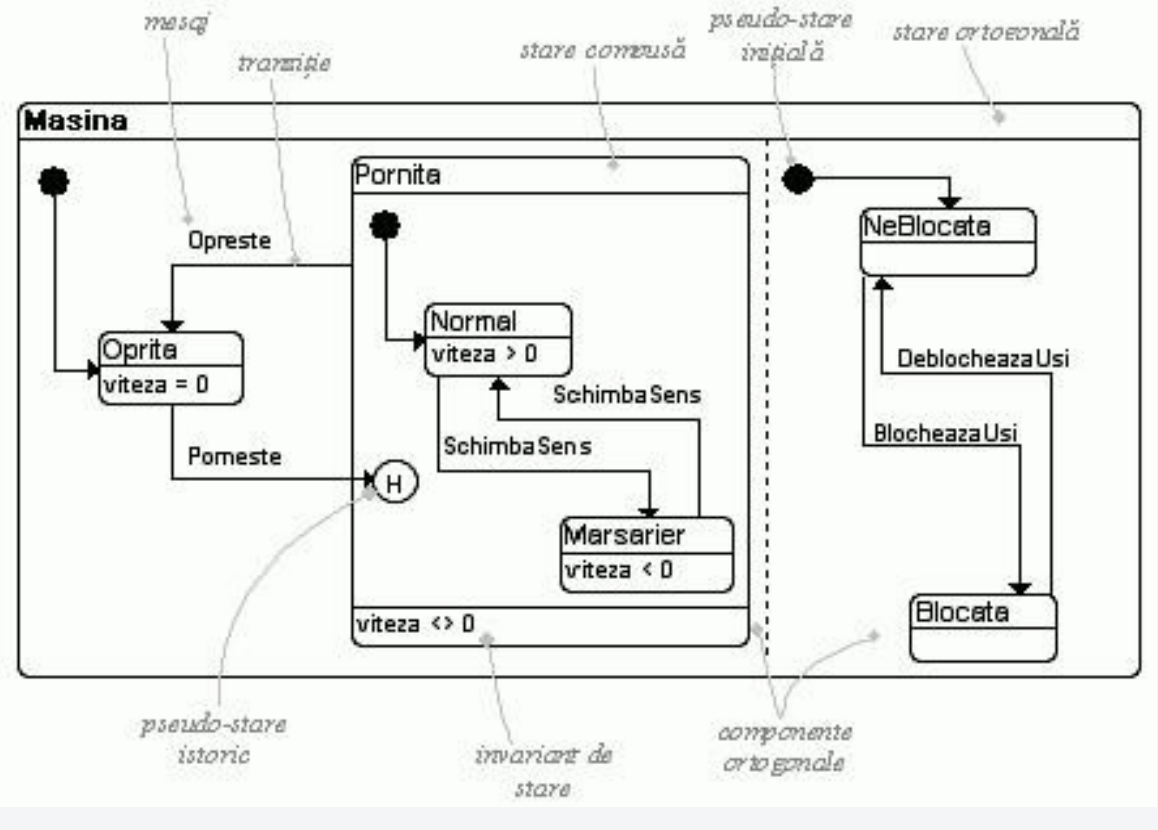
În UML diagramele de stări sunt utilizate în descrierea comportamentului obiectelor aparţinând unei clase.

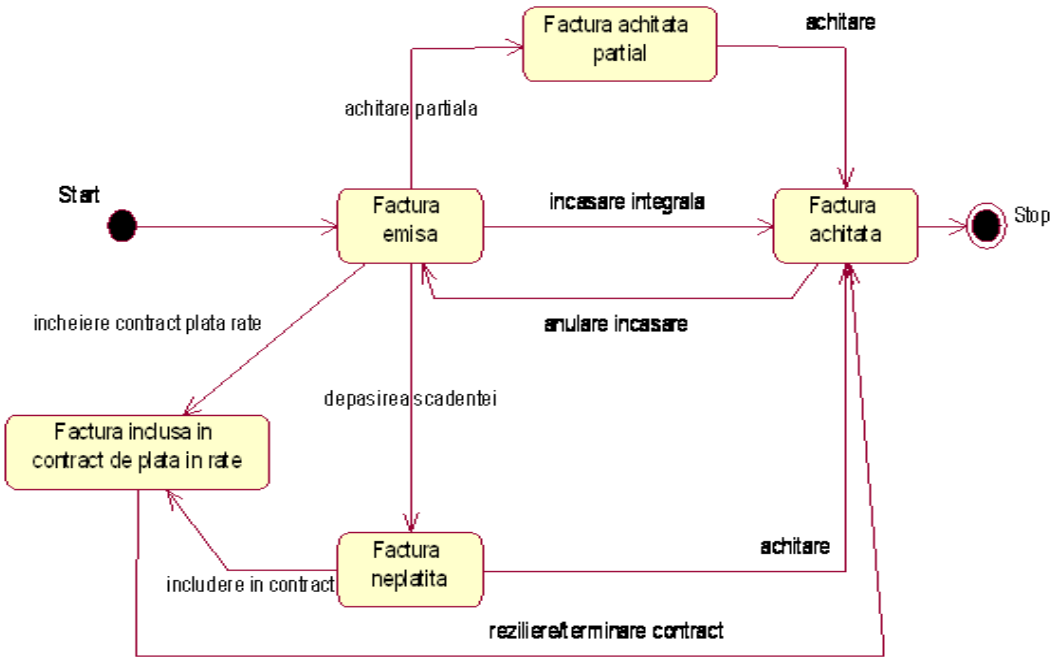
O stare (concretă) este caracterizată de valorile proprietăţilor unui obiect şi de mulţimea mesajelor care pot fi acceptate de către acest obiect la un moment dat. O stare conţine descrierea unui invariant de stare (condiţie logică adevărată pentru toate obiectele care se află în starea respectivă), şi a trei proceduri speciale: *entry*, *exit* şi *do*. Aceste proceduri descriu secvenţele de acţiuni care vor fi executate în momentul în care un obiect intră (*entry*), părăseşte (*exit*) sau se află (*do*) în starea respectivă. O stare se reprezintă grafic prin intermediul unui dreptunghi cu colţurile rotunjite, afişând în partea superioară un nume de stare. În partea inferioară a dreptunghiului opţional poate exista un compartiment care conţine expresiile ce definesc invariantul de stare şi cele trei proceduri speciale.

O tranziţie exprimă o situaţie în care un obiect poate trece dintr-o stare în alta. Tranziţiile se reprezintă grafic prin intermediul unor arce de cerc, linii simple sau linii poligonale orientate şi (opţional) etichetate care unesc două stări, numite stare sursă, respectiv destinaţie. Eticheta unei tranziţii este formată dintr-un mesaj, o condiţie (expresie logică) şi o secvenţă de activităţi care au loc în momentul declanşarii tranziţie. Pentru un obiect oarecare o tranziţie este declanşată atunci când obiectul se află în starea sursă a acesteia, execută operaţia corespunzătoare mesajului şi este îndeplinită condiţia specificată în etichetă.



Hărţile de stări permit modelarea de comportamente paralele ale unui obiect prin intermediul stărilor ortogonale (denumite şi AND-stări sau stări concurente). Stările ortogonale sunt formate din mai multe componente ortogonale, fiecare dintre acestea conţinând diverse sub-stări. Un obiect aflat într-o stare ortogonală se va afla de fapt în câte o stare corespunzătoare fiecărei componente ortogonale a acesteia.





2.4.4. Diagrame de colaborare

Diagramele de colaborare şi diagramele de secvență sunt reprezentații alternative pentru interacțiuni între obiecte.

O diagramă de colaborare este o diagramă de interacțiuni care arată secvențele de mesaj ce implementează o operație sau o tranzacție. O diagrama de colaborare prezintă obiectele, legăturile şi mesajele dintre ele. De asemenea, diagramele de colaborare pot conține simple instanțe de clase.

Fiecare diagrama de colaborare oferă o imagine asupra interacțiunilor sau a relațiilor structurale care au loc între obiecte şi a obiectelor ca entități în modelul curent.

2.4.5. Diagrame de secvență

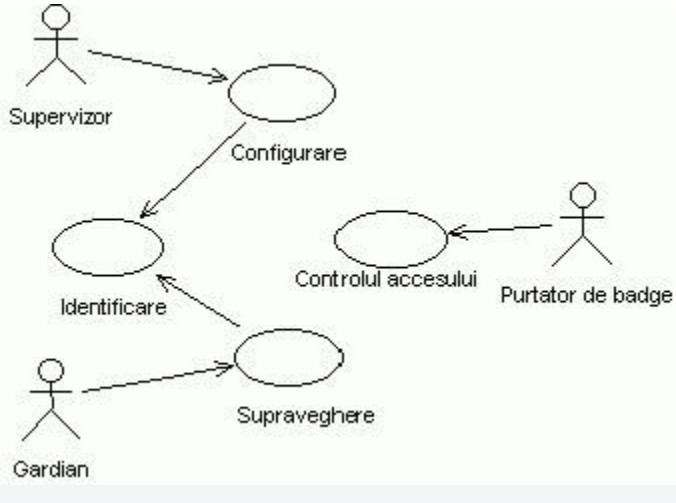
Diagramele de secvență prezintă interacțiunile între obiecte din punct de vedere temporal, contextul obiectelor nefiind reprezentat în mod explicit ca în diagramele de colaborare, reprezentarea concentrânduse pe exprimarea interacțiunilor.

O diagramă de secvență reprezintă o interacțiune între obiecte insistând pe cronologia (ordinea temporala) a expedieri mesajelor. Notația este derivata din diagramele MSC (Message Sequence Chart) ale grupului Siemens (1996).

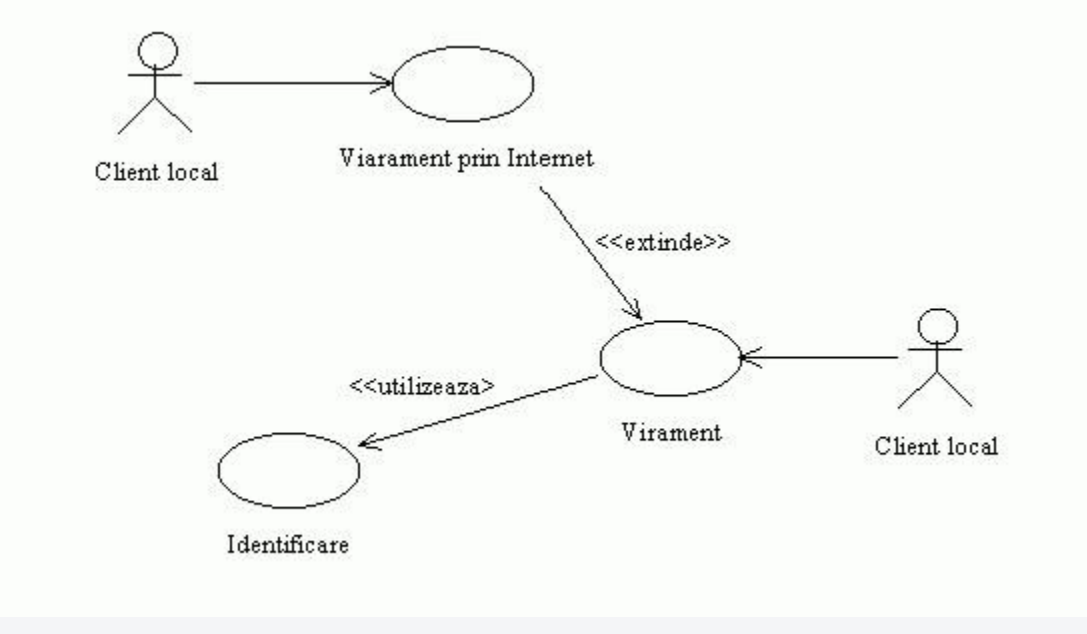
3. Anexe

3.1. *Diagrame use-case*

a. Diagrama cazurilor de utilizare pentru un sistem de control al accesului

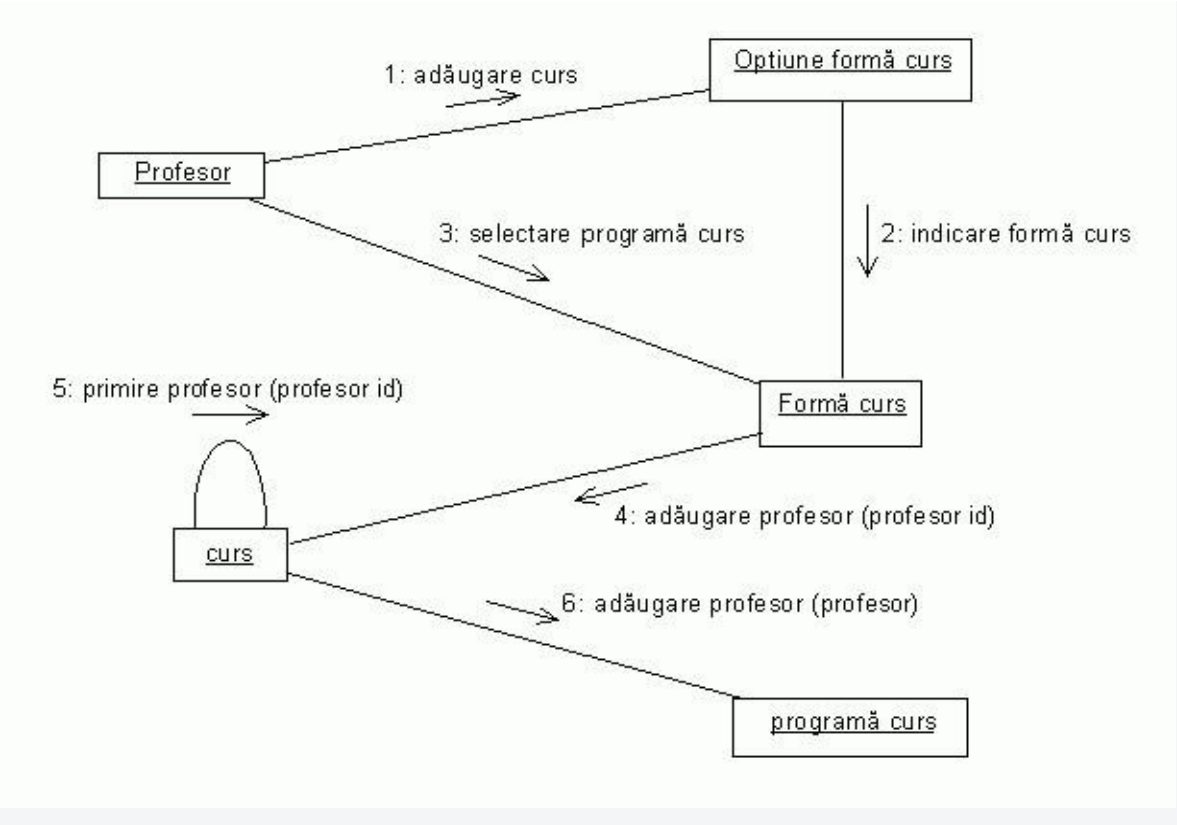


b. Diagrama cazurilor de utilizare pentru viramentul prin serviciul Internet



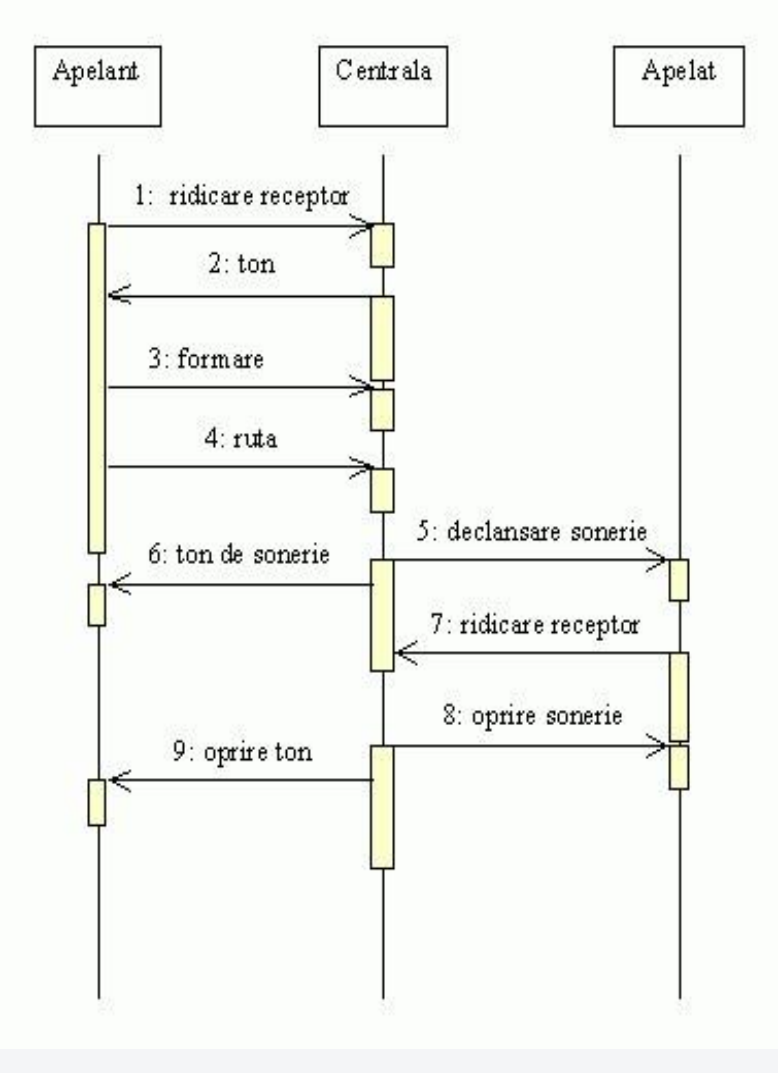
3.2. *Diagrame de colaborare*

Exemplu: Diagrama de secvență pentru cazul de utilizare a selectării unui curs:



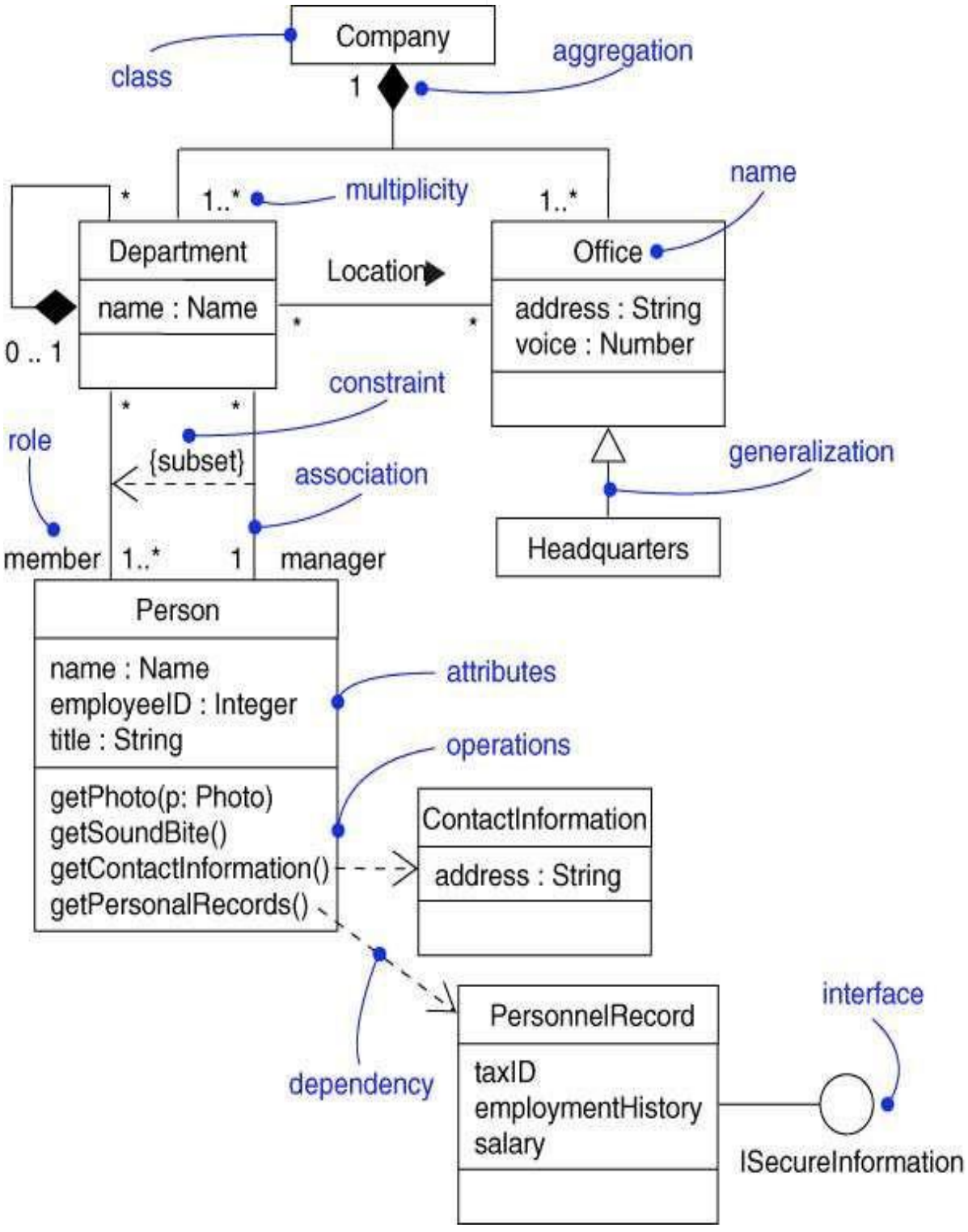
3.3. *Diagrama de secvență*

Exemplu: Diagrama de secvenţă pentru cazul de utilizare al unui apel telefonic



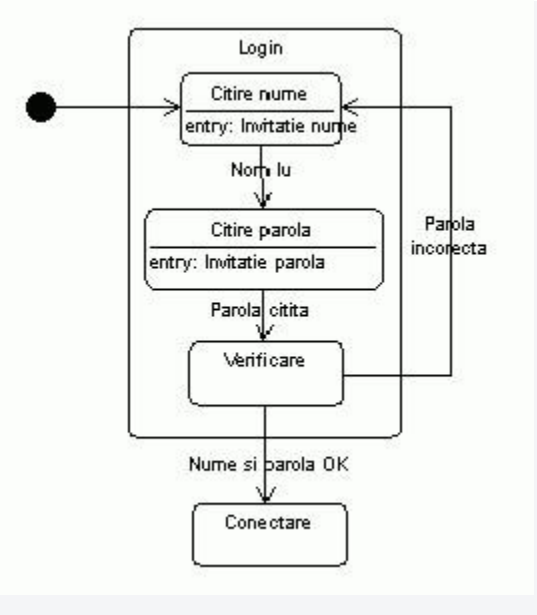
3.4. *Diagrame de clase*

Exemplu: Diagrama de clase pentru o companie


Company
class
aggregation
1
1..*
multiplicity
1..*
name
*
Department
name : Name
Location
Office
address : String
voice : Number
0 .. 1
*
*
*
*
constraint
role
{subset}
association
generalization
member
1..*
1
manager
Headquarters
Person
name : Name
employeeID : Integer
title : String
attributes
operations
getPhoto(p: Photo)
getSoundBite()
getContactInformation()
getPersonalRecords()
ContactInformation
address : String
PersonnelRecord
interface
dependency
taxID
employmentHistory
salary
ISecureInformation

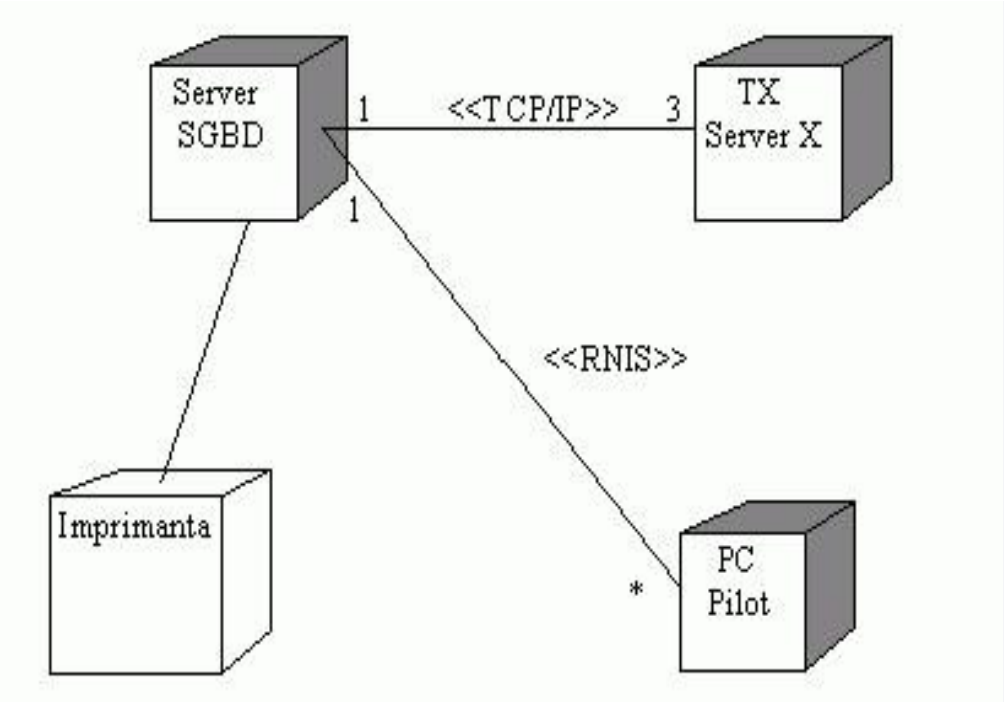
3.5. *Diagrama de stări*

Exemplu: Diagrama de stări pentru citire/verificare parola in cadrul unui sistem de control al accseului:



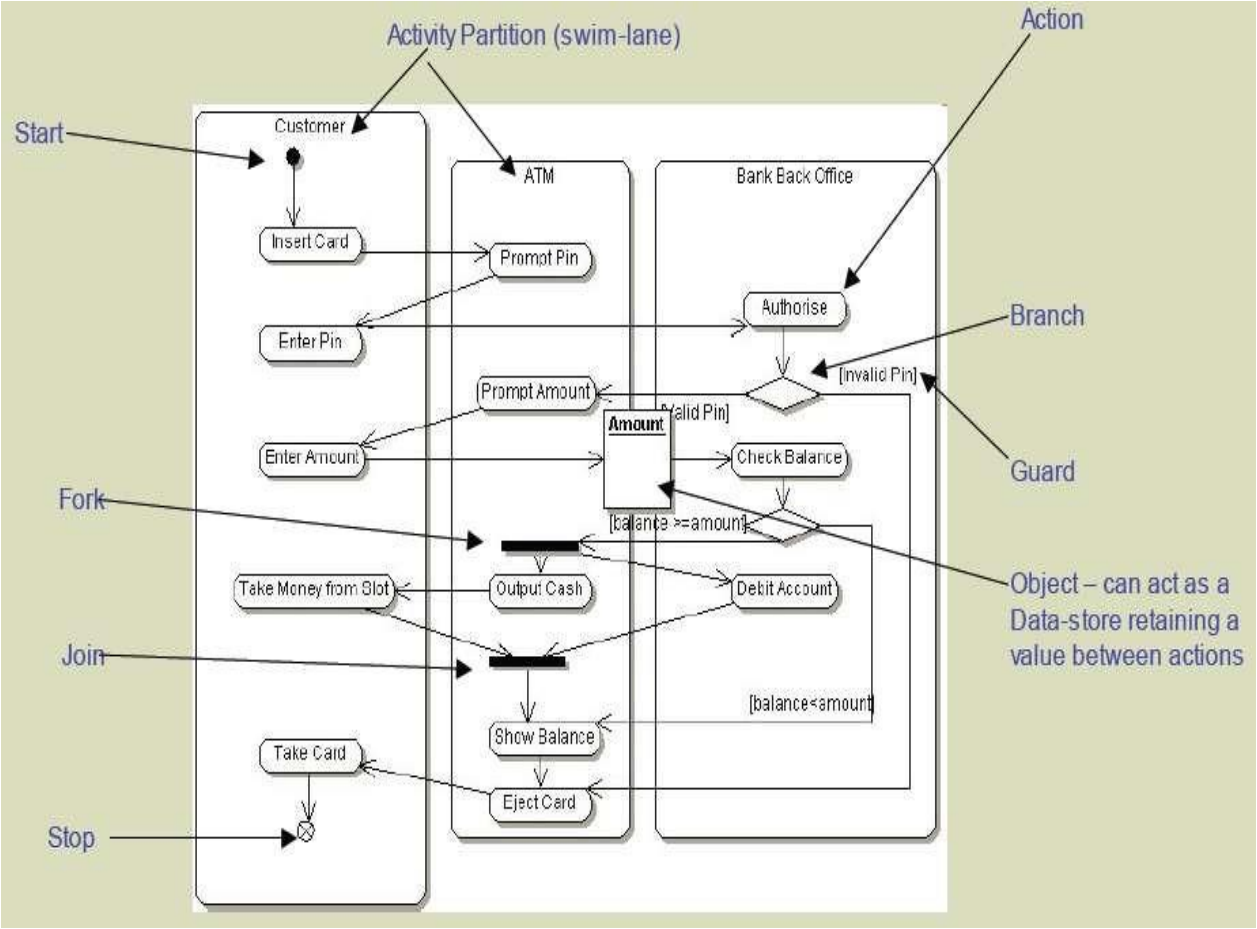
3.6. *Diagrame de implementare (deployment)*

Exemplu: Diagrama de implementare pentru o întreprindere



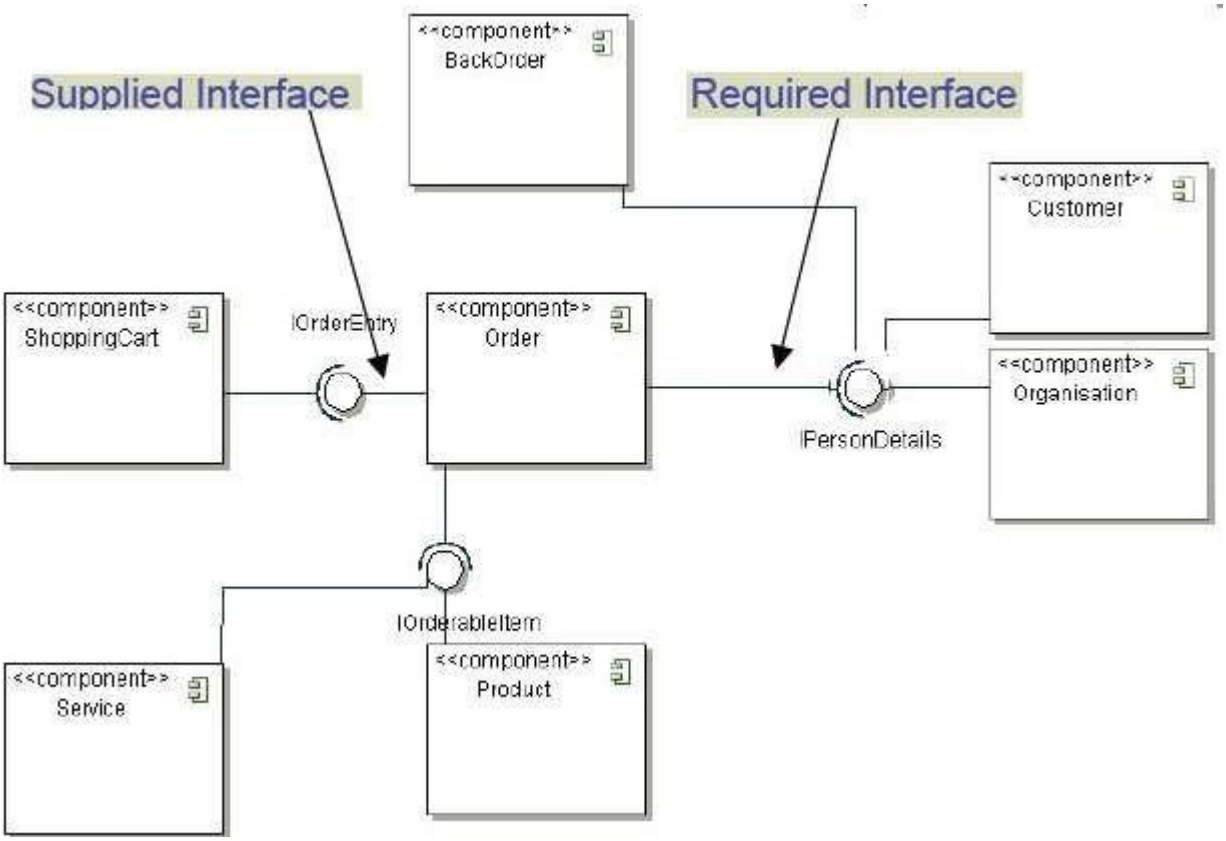
3.7. Diagrama de activitati

Exemplu: Diagrama de activitati pentru un ATM



3.8. Diagrama de componente

Exemplu: Diagrama de componente a unui magazin on-line



Bibliografie

1. Bennett, Simon; McRobb, Steve; Farmer, Ray, Object-oriented systems analysis and design using UML, ISBN 9781526849045, McGraw-Hill Higher Education, 2021
2. Rumpe, Bernhard, Modeling with UML: Language, Concepts, Methods,ISBN 978-3-319-33933-7, Springer 2016
3. Rumbaugh, James; Jacobson, Ivar; Booch, Grady, The Unified Modeling Language Reference Manual (2nd Edition), ISBN-13 9780321245625, Addison-Wesley Professional, 2004