Cuprins

Entităţi

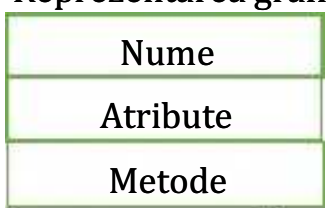
În UML sunt 4 tipuri de entităţi:

1. De structură
2. Comportament
3. Grupare
4. Adnotare

Entităţi de structură

Entităţile de structură sunt substantivele în UML. De regulă ele reprezintă părţile statice ale modelului care corespund elementelor conceptuale sau fizice ale sistemului. Entităţi de bază de structură:

1. **Clasa**. Clasa este descrierea multitudinii de obiecte cu aceleaşi atribute, metode şi sintaxe. Reprezentarea grafică în UML este un dreptunghi cu 3 secţii (nume, atribute, metode): Fig.



2. 1. Nivele de acces în clasă:
   public (+),
   private (-),
   protected (#).

3. **Interfaţa:**

   Interfaţa - reprezintă o totalitate de operaţii care definesc serviciile oferite de o clasă sau componentă (cerc, I Nume).

4. **Caz de utilizare:**

   Caz de utilizare - (elipsă orizontală cu linie neîntreruptă) reprezintă o descriere a consecutivităţii de acţiuni îndeplinite de sistem care produc un rezultat semnificativ pentru un anumit actor.

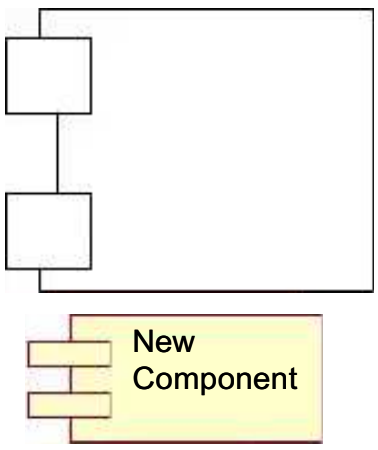
5. **Colaborare**:

   Colaborare - (elipsă orizontală cu linie întreruptă) defineşte o interacţiune care reprezintă o totalitate de roluri ce produc un efect corporativ.

6. **Clasă activă**:

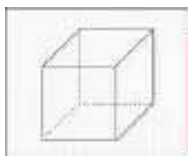
   Clasă activă - (clasă cu perimetru conturat) clasa elementele căreia sunt antrenate în unul sau mai multe procese şi pot iniţia o acţiune de administrare.

7. **Componenta:**



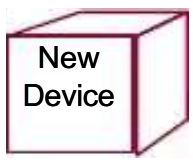
Componenta - (dreptunghi cu 2 bucle în stânga) reprezintă partea fizică a sistemului care corespund unui set de interfeţe şi asigură realizarea lui.

8. **Nod:**



- procesor şi dispozitiv în Enterprise

- dispozitiv numai în Rational Rose



Nod(cub) – partea reală a sistemului care conţine un mijloc de calcul cu un anumit volum de memorie şi deseori cu posibilitatea de prelucrare a informaţiilor. Există numai în timpul funcţionării unui program.

În afara acestor 7 unităţi mai sunt:

- actori, semnale, utilite – tipuri de clasă
- procese, şiruri – tipuri de clase active
- documente, tabele, aplicaţii, fişiere, biblioteci, pagini web – tipuri de componente

Entităţi de comportament

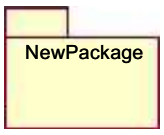
Entităţile de comportament sunt componentele dinamice ale UML, verbele limbajului care descriu comportamentul modelului în timp şi spaţiu. Sunt 2 tipuri:

1. **Interacţiunea:**
   I 
   Interacţiunea - (săgeată) – mod de comportare care constă în schimbul reciproc de mesaje între obiecte, în cadrul unui anumit context pentru atingerea unui scop anumit.

2. **Automat:**
   II 
   Automat - (dreptunghi orizontal cu colţuri rotunde) – algoritm care defineşte o succesiune de stări prin care trece obiectul pe parcursul ciclului de viaţă. Automatul e realizat de către o succesiune de elemente ca **stări** şi **tranziţii**.

Entităţi de grupare

a



Entităţile de grupare reprezintă părţile organizaţionale în UML. Există doar tipul **Package** (folder) – mecanism universal de organizare în grup. Poate grupa toate tipurile de entităţi.

Entităţi de adnotare

A

Entităţi de adnotare în UML se numesc părţi explicative. Este un tip **Note** (remark, filă – dreptunghi cu colţ îndoit) – reprezintă un simbol pentru prezentarea comentariilor şi restricţiilor. Legătura se face prin linie întreruptă.

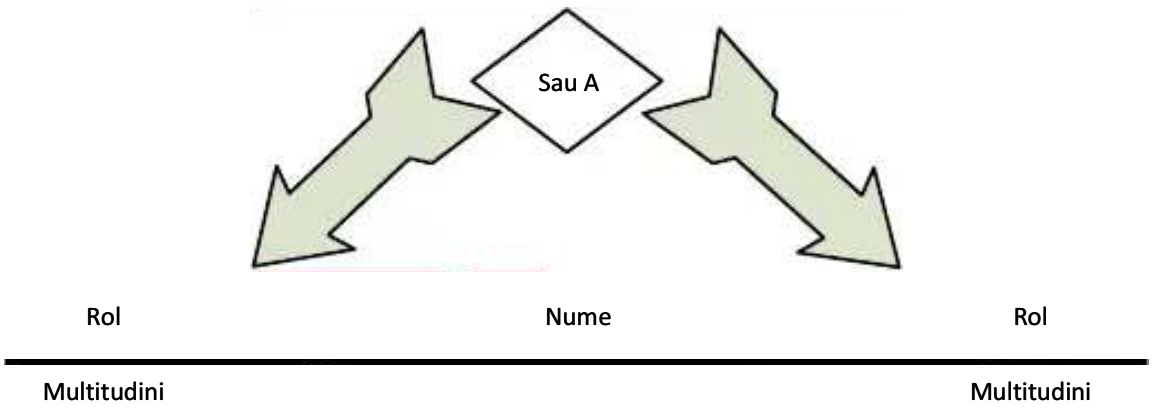
Relaţii în UML

În UML sunt 4 tipuri de relaţii:

1. Dependenţa
2. Asocierea
3. Generalizarea
4. Realizarea

Relaţiile sunt nişte elemente de legătură dintre entităţi.

1. **Dependenţa** – relaţia dintre 2 entităţi, una dintre ele fiind independentă iar cealaltă dependentă (dependent - - - - - → independent). Schimbarea stării celei independente va provoca schimbări în entitatea dependentă. Student - - - - → Profesor.
2. **Asocierea**



Asocierea – reprezintă o legătură semantică dintre 2 entităţi. Prin legătură se subînţelege schimbul reciproc de mesaje, … între obiecte. (Linie continuă, nume deasupra liniei pe centru, rol deasupra liniei stînga sau dreapta, multitudine sub linie stînga sau dreapta, triunghi haşurat spre stînga sau drepta indică cine primul trimite mesaje). Rol = Nivel_acces+Nume. Multitudine = {0, 0..1, 0..n, 1..n, n}. Asocierea poate să fie **unidirecţională** (linie continuă cu săgeată). Caz particular al asocierii este **agregarea**. La rîndul său agregare are o formă specială – **compoziţie**.

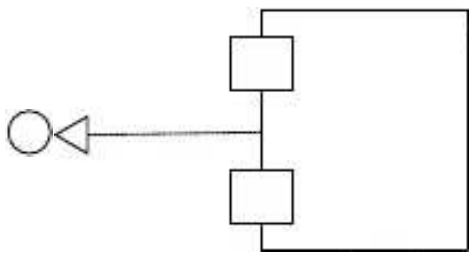
Agregarea (linie continuă cu romb nehaşurat la elementul întreg) – este o relaţie specifică dintre partea întreagă şi partea componentă.

Compoziţia

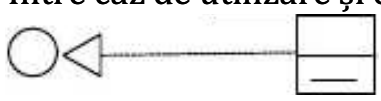
(linie continuă cu romb haşurat la elementul întreg) – relaţia dintre partea întreagă şi partea componentă, însă comparativ cu agregare, la compoziţie părţile componente nu pot exista fără partea întreagă.

3. **Generalizare**

(linie continuă cu triunghi nehaşurat la părinte) – relaţie de tip specializare/generalizare, în urma căreia elementul specializat (descendent) poate substitui elementul generalizat (părinte). Cu alte cuvinte descendentul moşteneşte structura şi comportamentul părintelui.
4. **Realizarea** (linie întreruptă cu triunghi nehaşurat) – relaţia dintre 2 entităţi una dintre care reprezintă garantul îndeplinirii unui „contract”, iar cea de-a doilea îndeplineşte contractul dat. Realizarea se utilizează în 2 cazuri:
    a. Între interfaţă şi clasa (sau componenta) ce o realizează



    b. Între caz de utilizare şi colaborarea ce o realizează



Diagramele în UML

1. Diagrama cazului de utilizare (Use Case)
2. Diagrame de interacţiune:
    a. Diagrama secvenţă
    b. Diagrama colaborare
3. Diagrama claselor
4. Diagrame de comportament:
    a. Diagrama de stări
    b. Diagrama de activităţi
5. Diagrama de componente
6. Diagrama de amplasare

Diagrama cazului de utilizare

Modelarea vizuală în UML poate fi reprezentată ca un proces de lansare pe nivel de la cel mai general (abstract) spre modelul logic, apoi spre modelul fizic.

Diagrama cazului de utilizare (DCU) descrie destinaţia funcţională a sistemului (ceea ce sistemul va executa în procesul său de funcţionare).

Proiectarea DCU are scopurile:

- Determinarea limitelor comune şi a contextului domeniului de modelare la etapa iniţială de proiectare a unui sistem
- Formularea cerinţelor comune către comportarea funcţională a sistemului proiectat
- Elaborarea modelului iniţial, conceptual al unui sistem pentru următoarea detaliere în formele logice şi fizice ale sistemului

În caz general DCU reprezintă un graf, entităţile cărora (actorii) sunt muchiile şi relaţiile sunt legăturile.

Elementele DCU

Caz de utilizare

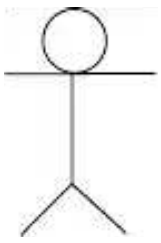
Elipsă cu linie neîntreruptă.

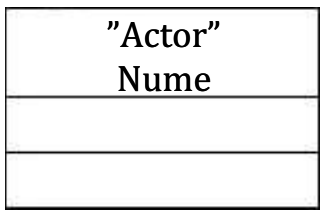
Cazul de utilizare se utilizează pentru particularităţile comune ale comportării unui sistem sau a oricărei alte entităţi din domeniul de lucru, fără de a cerceta structura internă a acestei entităţi. Fiecare caz de utilizare determină o succesiune de acţiuni care trebuie executate de un anumit actor.

Numele cazului de utilizare trebuie să fie un verb sau frază verbală.

Scopul principal a cazului de utilizare constă în determinarea comportării unei anumite entităţi fără descrierea structurii interne a acesteia. Fiecare caz de utilizare corespunde unui serviciu (actor) aparte.

Actor





Grafic: un celovecic, omuleţ.

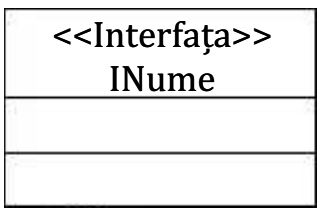
Actorul reprezintă o entitate externă faţă de sistemul proiectat care interacţionează cu sistemul şi utilizează posibilităţile lui funcţionale pentru atingerea unor anumite scopuri.

Actorul poate fi: om, sistem, instalaţie tehnică, produs program, … care pot fi sursă de acţiune pentru sistemul proiectat.

În unele cazuri poate fi reprezentat cu o clasă cu stereotipul «actor».

* Substantivele care sunt plasate între ghilimele „__" sau «__» în UML se numesc stereotipuri.

Interfaţa



Cerculeţ mic. Numele: I+Nume.

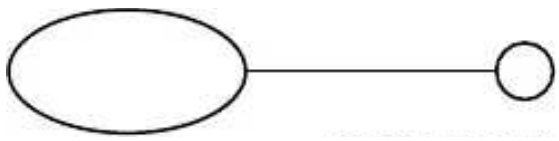
Interfaţa specifică parametrii modelului care sunt vizibili dinafara sistemului fără de a ne interesa structura lui internă. Pentru DCU interfaţa defineşte o totalitate de operaţii care asigură serviciile necesare sau funcţionalitate pentru careva actori.

Interfaţa nu poate conţine nici atribute, nici stări, nici asocieri dirijate. Ele conţin doar operaţii fără indicarea specificaţiilor de realizare a lor. Cu alte cuvinte interfaţa reprezintă cu o clasă abstractă.

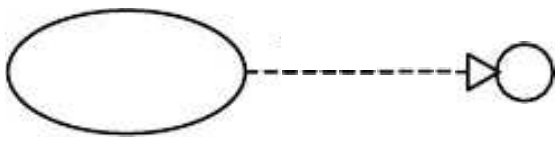
Interfaţa de asemenea poate fi reprezentată ca clasă cu stereotipul «interface».

Se utilizează în 2 cazuri:

1. IFormăComandă – procurare produs. Fig



2. IDateClient – Identificarea clientului. Fig



Colaborare

Vezi entitatea de structură

Relaţii în diagrama cazului de utilizare

1. Un anumit actor poate colabora cu mai multe cazuri de utilizare, în acest caz actorul nu face altceva decât se adresează la careva servicii ale sistemului dat. La rândul său un caz de utilizare poate colabora cu mai mulţi actori(oferă serviciile sale).
2. * Două cazuri de utilizare definite pentru aceeaşi entitate nu poate colabora între ele(nu poate fi relaţii de asociere). Fiecare din ele au o variantă proprie determinată pentru o anumită entitate. În diagrama caz de utilizare sunt 4 relaţii de bază:
   a) Asocierea.
   b) Extinderea.
   c) Generalizarea.
   d) Includerea.

I. Asocierea.

Pentru diagrama caz de utilizare relaţia de asociere specifică rolurile unui actor pentru situaţii concrete.

Fig.1.

Dacă la relaţii de asociere nu este indicată multitudinea, implicit este 1 (unu).

II. Extinderea.

Relaţia de extindere defineşte interconexiunea a unui caz de utilizare cu caz de utilizare general, proprietăţile cărora sunt definite în baza modelului de uniune a exemplarelor date(unde se intersectează). În meta-modelul UML relaţia de extindere este directă şi indică care condiţii pot fi utilizate pentru a extinde caz de utilizare general.

Relaţiile de extinderea şi include nu sunt altceva decât nişte stereotipuri a relaţiilor de independenţă.

Fig.2

III. Generalizarea.

Relaţiile de generalizare sunt pentru indicarea faptului că un caz de utilizare:

A Poate fi generalizat un caz de utilizare. În urma căruia **A** va deveni caz special **B**, sau cu alte cuvinte **B** este părinte pentru **A**.

Fig.3

IV. Includerea.

Relaţia de includere … care indică comportamentul stabilit pentru un caz de utilizare care este inclus compus în alt caz de utilizare.

Fig.4
Trace este un **include istoric (ceea ce a fost făcut deja).**

Fig.5

Diagrama de interacţiune.

Diagrama de secvenţă.

În limbajul UML colaborarea între obiecte se cercetează în aspectul informativ al comunicaţiilor lor, cu alte cuvinte obiectele ce interacţionează nu fac altceva decât un simplu schimb de mesaje dintre ele. În UML sunt 2 tipuri de diagrame de interacţiune:

1. Obiectele colaborează între ele în timp şi atunci pentru prezentarea particularităţilor temporale se utilizează diagrama de secvenţă.
2. Pot fi cercetate particularităţile structurale ale colaborărilor între obiecte (schimbul de mesaje în spaţiu).

Elementul principal din diagrama de secvenţă este obiectul.

Obiectul este o instanţă a clasei. Obiectul în diagrama de interacţiune se reprezintă în formă de dreptunghi.

Fig.6.

Numele obiectul poate conţine: nume obiect =O, tip clasă=C, tip pachet=P (N_o:T_c::T_p)

Nume obiect nu poate fi fără nume clasă.

1) O:C::P
2) O:C
3) O: - obiect orfan (când : sunt puse dar nu este specificată clasa)
4) O
5) :C::P – obiect anonim
6) :C – obiect anonim al clasei C

Fiecare obiect are o **linie verticală punctată**, linia dată indică timpul de viaţă a obiectului în sistemul dat. **Elementul X** semnifică că obiectul a fost distrus. **Focus control** timpul când obiectul este activ. Obiect în diagrama de secvenţă apar în dependenţă de interacţiune.

Rational Rose

1. Relaţie generală.
2. R. Sinhronă.
3. R. Asinhronă.
4. R. Timeout
5. R. Balking.
6. R. Chemarea procedurii

7. R. Return.

Enterprise Arhitect

I. Sinhronă
II. Asinhronă
III. Return.

Stereotipuri

a) Call – invocă o operaţie sau procedură obiectului destinatar.
b) Return – returnează valoarea operaţiei executate obiectului apelant.
c) Create – creează un anumit obiect pentru careva interacţiune.
d) Destroy – distruge obiectul pentru eliberarea memorie.
e) Send – expedierea unui semnal (mesaj).

Rational Rose utilizează:

1. Utilizează: a,b,c,d,e.
2. Utilizează: a,b,c,d,e.
3. Utilizează: a,b,c,d,e.
4. Utilizează: a,b,d,e.
5. Utilizează: a,b,d,e.
6. Utilizează: b,c,d.
7. Utilizează: a,d,e.

Enterprise Arhitect utilizează:

I. Utilizează: b,c,d.
II. Utilizează: a,b.
III. Utilizează: a,d,e.

- În limbajul UML informaţia care este scrisă în acolade {} se numeşte restricţie de timp.
- **End – Diagrama de secvenţă.**

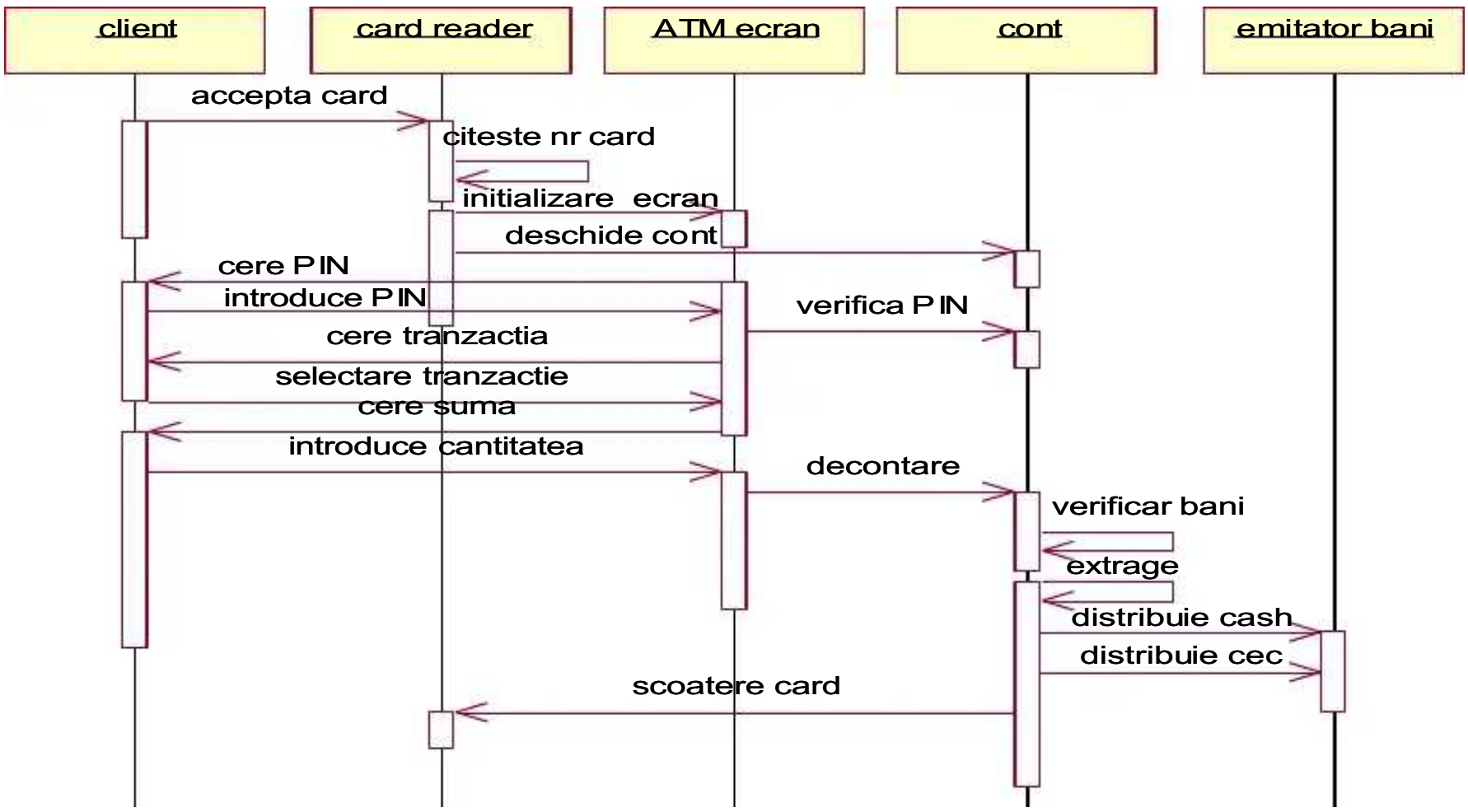


Fig.1.5. Diagrama de secvenţă

Diagrama de colaborare

Particularitatea principală constă în reprezentarea grafică nu numai a interacţiunii dintre obiecte ci şi relaţiile structurale dintre obiecte.

Spre deosebire de diagrama de secvenţă în diagrama de colaborare se reprezintă relaţiile între obiecte care au o oarecare semnificaţie în timpul colaborării.

În diagrama de colaborare nu se indică timpul interacţiunii obiectelor. Însă în diagrama de colaborare există posibilitatea de a numerota mesajele (interacţiunile), ceea ce ne permite de a vedea schimbul de mesaje în timp.

Ca și în diagrama de secvență, principalul element este obiectul (instanţa unei clase). Numele obiectului poate fi: Nume_ob/Rol_clasă:Tip_clasă::Tip_pachet. Exemple:

- `O/R:C::P` – Obiectul O cu rolul R, clasa C din pachetul P
- `/R:C::P` – Obiectul anonim O cu rolul R, clasa C din pachetul P
- `:C::P`
- `:C`
- `O :C::P`
- `O :C`
- `O`
- `O :` – Obiect orfan
- `/R:C`
- `/R`
- `O/R`

- O/R:

Multiobiect

Multiobiect reprezintă o mulţime de obiecte la una din terminaţiile asocierii.

În diagrama de colaborare entitatea multiobiect se utilizează pentru a reprezenta operaţiile şi semnalele care sunt adresate către multitudini sau totalităţi de obiecte. Multiobiectul se reprezintă grafic printr-un dreptunghi în alt dreptunghi (Fig 1).

***Relaţia dintre multiobiect şi obiectul care face parte din el – compoziţia.** (Fig. 2)

Asociere (Fig 3)

Obiecte active şi pasive

În UML obiectele se împart în active şi pasive.

Obiectele **pasive** folosesc numai datele care sunt transmise şi nu pot avea activitate de control. Însă ele pot transmite semnale pe parcursul timpului de realizare a cererii.

Obiect **activ** are un flux de control propriu şi poate iniţializa activitate de control. (Fig 4)

Obiect compus

Obiectul compus se mai numeşte obiect container. El este destinat pentru reprezentarea obiectului care are structură proprie şi flux intern de control.

Obiectul compus este exemplarul unei clase compuse care este legat cu părţile sale componente prin relaţia de **agregare** sau **compoziţie**.

Obiectul compus se reprezintă prin dreptunghi din 2 componente: numele obiectului şi numele obiectelor interne (Fig 5).

Stereotipuri de legături

- „association” (FIG 6) – asociere, stereotip implicit
- „parameter” – parametrul metodei. Obiectul cărui i s-a transmis stereotipul parameter poate fi doar parametru al unei metode
- „local” (Fig 7) – variabilă locală a metodei. Nivelul de vizibilitate a variabilei locale e limitat pînă la obiectul vecin.
- „global” – variabilă globală a metodei. Vizibil pentru toate elementele diagramei de colaborare.
- „self” (Fig 8) – relaţie recursivă.

Elementul colaborare

Noţiunea de colaborare înseamnă o mulţime de obiecte care interacţionează în context comun. Scopul colaborării constă în specificarea particularităţilor realizării celor mai semnificative operaţii în sistem.

Niveluri

Diagrama de colaborare poate fi reprezentată în 2 nivele:

1. Nivel de specificare – se reprezintă rolul clasificatoarelor şi rolurile asocierilor în colaborarea dată
2. Nivel de exemplu – diagrama reprezintă obiectele şi legăturile dintre ele cu mesaje

Diagrama de colaborare de nivel de specificare reprezintă rolurile elementelor (nivel de clase) ce participă la colaborare. La nivel de exemple sunt reprezentate doar obiectele care au legătură cu realizarea operaţiilor.

(Fig 9) – nivel de specificare

(Fig 10) – Colaborare nivel de exemplu

(Fig 11) – Diagrama claselor

Diagrama claselor

Diagrama claselor se utilizează pentru reprezentarea structurii statice ale sistemului în terminologia POO. În diagrama claselor pot fi reflectate diferite legături dintre entităţi (clasa, interfaţa, pachetul, obiectul) şi poate descrie structura lor internă. Bineînţeles că pot fi utilizate toate 4 tipuri de relaţii UML.

În diagrama de clase nu poate fi reprezentat aspectul temporar al funcţionării sistemului. Diagrama de clase nu e altceva decât dezvoltarea de mai apoi a diagramei de structură.

Diagrama claselor se reprezintă ca orice altă diagramă sub formă de graf. Vîrfurile sunt entităţile, muchiile – relaţiile.

Clasa

Clasa – dreptunghi cu 3 părţi. Prima – numele, a doua – atributul (Nivel de acces Nume atribut [multiplicitate] : Tip valoare = Valoare iniţială sau {constantă}), a treia – metodele (Nivel de acces Nume metodă (lista parametrilor) : Tip valoare întoarsă).

Relaţii

Sunt 4 tipuri de **relaţii** în diagrama claselor:

- Dependența – relația dintre 2 entități (clase), una independentă (cu săgeata spre ea) și cealaltă – dependentă. Stereotipuri:
  - «access» - servește ca indicator de accesibilitate pentru atributele și operațiile clasei sursă pentru clasa client
  - «bind» - clasa client (dependentă) poate utiliza șabloane pentru următoarea parametrizare a sa
  - «derive» - atributele clasei client pot fi calculate după atributele clasei sursă
  - «import» - atribute și metode publice ale clase sursă pot fi considerate ca atribute și metode ale clasei client
  - «refine» - indică că clasa client servește ca precizie pentru clasa sursă (ca caracter istoric)
- Asocierea – legătură semantică dintre 2 clase. Rol = nivel acces (+, -, #) + nume rol. Triunghi hașurat indică cine are primul acces la obiect. Multitudini (0, 0..1, 1, 0..*, 1..*, *, 7, 2..5) – standard e 1.
- Generalizarea – relație semantică dintre elementul generalizat … Linie neîntreruptă cu triunghi nehașurat. Restricțiile între paranteze figurate:
  - {complete} – în diagramă sunt reprezentate toate clasele descendente, iar altele nu mai pot exista
  - {incomplete} – nu sunt reprezentate toate clasele descendente
  - {disjoin} – clasele descendente nu pot conține obiecte care concomitent sunt exemplare a 2 sau mai multor clase
  - {overlapping} – moștenire multiplă (figura moștenește clasa punct și clasa linie ☺)
- Realizarea – clasa realizează interfața (linie întreruptă cu triunghi la interfață

Interfaţa (vezi …)

Obiectul (vezi …)

Figură: Diagrama de clase Universitate

Diagrama de stare

Pentru modelarea comportamentului la nivel logic în UML pot fi folosite următoarele diagrame:

- Stare
- Activităţi
- Secvenţă
- Colaborare

Spre deosebire de alte diagrame, diagrama de **stare** descrie procesul de modificare a stărilor

pentru o anumită clasă (pentru obiectul, exemplarul unei clase), cu alte cuvinte reprezintă procesul de modificare a stărilor a unui anumit obiect. Schimbarea stării obiectului poate fi provocată de alte obiecte din exterior sau din interior.

Diagrama dată se utilizează pentru descrierea consecutivităţii de stări posibile şi treceri care împreună caracterizează comportamentul obiectului pe perioada ciclului său de viaţă.

Automat

Automatul în UML reprezintă o formalitate pentru modelarea comportamentului modelului a unui sistem întreg. Automatul este un pachet în care putem defini o mulţime de definiţii care sunt necesare pentru a reprezenta comportamentul entităţii modelate.

Automatul – diagramă de stare.

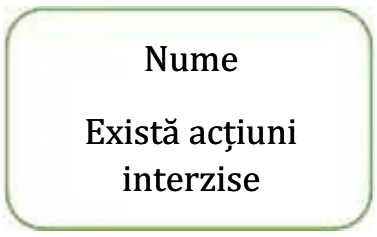
Starea

Noţiune de bază care intră în formalismul automatul sunt stările şi tranziţiile (trecere). Diferenţa prihcipală dintre stare şi tranziţie constă în:

- Durata aflării sistemului în stare depăşeşte cu mult timpul care este destinat trecerii. Se presupune că timpul tranziţiei este 0 şi se trece dintr-o stare în alta momentan

➢ În limbajul UML sub-stare se subînţelege o meta-clasă abstractă care se utilizează pentru modelarea situaţiilor concrete. Pe parcursul căruia sunt prezente careva acţiuni, din cadrul clasei cât şi în afară clasei. Starea poate fi în formă de valori concrete, a atributului sau a obiectului clasei, în cazul acestuia modificarea a unor valori va duce spre modificarea stării obiectului.

E de menţionat că nu fiecare atribut a clasei poate să caracterizeze starea obiectului. Starea în UML se reprezintă grafic în formă de dreptunghi cu colţurile rotunjite.



În afară de nume starea poate avea şi acţiuni interne.

Fig.1.

Numele stării

Numele stării nu este altceva decât un text care dezvăluie sensul stării date de regulă numele se scrie cu majusculă, verbul la timpul prezent sau participiu. În unele cazuri numele stării poate lipsi în acest caz starea este anonimă.

Dacă în diagramă sunt mai multe stări anonime trebuie să redefinim între ele. În diagrama de stări

putem utiliza şi pseudo stări. Pseudo stări sunt de 2 tipuri:

1. Pseudo stare iniţială.
2. Pseudo stare finală.

Ambele sunt cazuri particulare a ale stării care nu conţin nici o acţiune internă. Pseudo stare iniţială se utilizează pentru a ne reprezenta momentul de la care începe modificarea stărilor obiectului. Pseudo starea finală ne reprezintă sfârşitul de reprezentare a stării sau sfârşitul ciclului de viaţă a obiectului.

Fig.2.

În diagrama de stare poate fi doar o singură pseudo stare iniţială şi din ea iese doar o singură tranziţie. Iar pseudo stări finale pot fi 1,2 şi mai multe în dependenţă de caz şi în ea doar intră.

În cazul când avem mai multe pseudo stări finale este obligatoriu de a le diferenţia prin atribuirea diferitor nume.

Tranziţia

Trecerea dintr-o stare în alta.

Tranziţia – reprezintă o relaţie dintre 2 stări consecutive ceea ce ne indică trecerea de la o stare la alta. Ea poate fi de 2 tipuri:

1. Trigher .
2. Netrigher

Dacă trecerea nu are suplimentar careva aliniat de text (de regulă eveniment) atunci astfel de tranziţie se numeşte *netrigher*, în rest *trigher*.

Signatura_evenimentului[condiţia_gardă]/acţiunea
Signatura_evenimentului ----
Condiţia_gardă - se verifică dacă este true sau false.

În cazul când sunt mai multe condiţii gardă se despart prin **; (punct şi virgulă).**

Starea compusă

Starea compusă - este starea compusă care este alcătuită din 2 sau mai multe stări depuse. Stările depuse vor fi sub-stări pentru starea iniţială. Relaţia dintre sub-stări şi starea iniţială va fi compoziţia.

Fig.3.

Sub-stări disjuncte

Sub-stări disjuncte – se utilizează pentru modelarea comportamentul obiectului în timpul căruia obiectul se poate afla doar într-o singură sub-stare.

Fig.4.

Sub-stare compusă cu stări concurente.

Sub-stări concurente

Sub-stări concurente – pot specifica 2 sau mai multe sub-automate care pot executa în paralel sau paralel. Fiecare sub-automat are regiunea sa în starea compusă. Sub-automatul poate fi atât stare compusă cu sub-stări depuse cât şi starea compusă cu stări disjuncte.

Fig.5.

Exemple diagrama de stare

Fig. 6

Diagrama de activităţi

Pentru modelarea proceselor de executare a operaţiilor în limbajul UML se utilizează diagrama de activităţi. Diagrama de activităţi poate fi numită ca un caz particular diagrama de stări (are aceleaşi tipuri de activităţi şi elemente). Fiecare stare de activitate corespunde unei operaţii elementare şi ca şi în cazul diagramei de stări trece în diagrama de activitate după executarea diagramei de stări.

Sensul principal al acestei diagrame constă în vizualizarea particularităţilor , operaţiilor al unor clase pentru a reprezenta algoritmul de execuţie. Diagrama de activitate - este un caz particular a stării însă nu poate avea tranziţii interne.

Grafic se reprezintă printr-un dreptunghi unde are prin părţi 2 bucle.

Nume
a:=b+c

Uneori este necesar de a menţiona că starea de activitate este o stare compusă.

Diagrama de tranziţie - în comparaţie cu diagrama de stări toate tranziţiile sunt **netrigher.**

Fig.7

Dintr-o diagramă de stare poate fi doar o intrare/ieşire.

În diagrama de activităţi este o pseudo-stare iniţială şi o pseudo-stare finală.
Fig.8

Cel mai mare neajuns al schemelor bloc constă în aceea că nu putem reprezenta stări paralele. În UML există elemente speciale pentru reprezentarea calculelor paralele, acest simbol se reprezintă prin linie dreaptă, verticală sau orizontală, analogic notaţii unei tranziţii în formalismul reţelelor Petre. Sunt 2 entităţi fork şi join.

1. Fork – divizarea în mai multe fluxuri paralele, deci are 1 intrare şi 2 ieşiri.
2. Join – unirea fluxelor paralele, mai multe intrări şi 1 ieşire.

Fig.9

....

Fig.10

Partiţia

Diagrama de activităţi poate fi folosită nu numai pentru specificarea algoritmilor de calcul sau a fluxului de control a sistemelor de programare, dar şi în modelarea business proceselor. Activitatea oricărei companii reprezintă o totalitate de acţiuni independente care sunt îndreptate spre un anumit rezultat.

Modelarea business proceselor la se utilizează pentru reprezentarea subdivizării.

Subdivizarea sunt responsabile de careva acţiuni iar business procesele reprezintă trecerea de la o subdiviziune la alta. Pentru modelarea aş tipuri de particularităţi în limbajul UML se utilizează partiţiile.

Partiţiile sunt divizate prin linii verticale neîntrerupte. 2(două) linii verticale reprezintă partiţia sau subdiviziunea.

Linia partiţiilor pot fi întretăiate doar de tranziţie.

Obiectul

Obiectul în afară de nume poate conţine şi careva caracteristici (se scriu după numele obiectului în paranteze pătrate []). Între obiect şi orişicare element dintre diagramă poate fi utilizat relaţia de dependenţă (linie neîntreruptă cu săgeată).

Obiectele pot fi plasate cât şi în cadrul partiţiei cât şi pe linia partiţie.

Diagrama componente

Toate diagramele analizate până acum se refereau la aspectele conceptuale de proiectare a unui model de sistem şi se refereau la nivel logic de reprezentare. Specificul reprezentării logice constă în faptul că el utilizează noţiuni ce nu au personificare personală(materială). Diferite elemente are reprezentări logice (clase, obiecte, mesaje, relaţii, etc) nu există în mod material sau fizic, ele reflectă doar înţelegerea noastră despre sistemul fizic. Destinaţia principală a reprezentării logice constă în analiza relaţiilor structurale şi funcţionale între elementele unui model. Pentru reprezentarea elementelor la nivel fizic este necesar de a transforma toate entităţile logice în entităţi materiale. În limbajul UML pentru reprezentarea nivelului fizic se utilizează diagrama de componente şi de amplasare.

Diagrama de componente permite determinarea arhitecturii sistemului elaborat prin determinarea dependenţilor între componente care pot fi (fişiere text, fişiere cu cod sursă, biblioteci, fişiere dinamice, etc.). În majoritatea cazurilor componenta corespund unui anumit fişier. Diagrama de componente se utilizează pentru următoarele scopuri:

1. Vizualizarea structurii comune a codului sursă a careva aplicaţii.
2. Specificarea variabilei executabile a careva aplicaţii.
3. Reutilizarea cărorva fragmente din cod sursă.
4. Reprezentarea conceptuală şi fizică a sistemelor de baze de date.

Elementele principale a diagramei de componente sunt:

1. Componenta
2. Interfaţa (entităţi)
3. Dependenţa (relaţiile)

Componenta

Pentru reprezentarea entităţilor fizice în limbajul UML se utilizează componenta, ea poate realiza un set de interfeţe şi desemnează elementele reprezentării fizice a unui model.

Fig.1 & Fig.2

Componentele pot fi reprezentate la nivel de tip sau la nivel de exemplar.

La nivel de tip: este **fig.1 şi fig. 2**

La nivel de exemplar se notează -
numele component: numele exemplar

Fig.3

Dacă este exemplar trebuie să aibă linia de jos.

Tipuri de componente

Fişiere cu extensia *.dll

Fig.4

Fig.5

Aceste notaţii adăugătoare nu sunt considerate ca elemente de bază a limbajului UML însă utilizarea lor uşurează înţelegerea diagramelor.

În diagrama de componente sunt specificate 3 feluri de componente:

1. **Componente de regrupare** - care specifică executarea de către sistem a fucţiilor sale, aşa componente pot fi: biblioteci dinamice (*.dll), web pagini (html, xml, php), fişierele help (*.hlp).
2. **Produse de lucru** – pentru limbajul c++ aceste sunt fişierele cu extensia *.h şi *.cpp.
3. **Componente de executare** – aceste elemente în unele cazuri mai sunt numite artefacte (în aşa fel se subliniază conţinutul lor informaţional).

Un alt mod de specificare a tipurilor de componente este indicarea stereotipurilor componentei care de regulă este plasat deasupra numelui.

Stereotipuri:

1. Library – se referă la primul tip de componente (de regrupare) şi de obicei indică sau reprezintă bibliotecile dinamice sau statice.
2. Table – se referă la primul tip de componente (de regrupare) care reprezintă tabele bazei de date.
3. File – se referă la al doilea tip de componente care de regulă reprezintă fişierele cu texturl iniţial a programului.
4. Document - se referă la al doilea tip de componente produs de lucru şi reprezintă un document.
5. Executable - se referă la al treilea tip de componente.

Interfata

Vezi entitatea de structură

Fig.7

Dependența

Este considerată relaţia principală