Diagrame de clase conceptuale în limbajul de modelare unificat (UML)

prof. dr. ing. Mircea Petrescu

UML = limbaj de modelare vizual, orientat obiect, care descrie (reprezintă) proprietățile structurale şi dinamice ale unui sistem software. Prin sistem software se înțelege o BD sau un modul de cod în general.

Spre deosebire de modelul EAE, *UML* este o *colecție de tehnici de modelare*, folosite pentru tratarea multor aspecte ale procesului de concepere şi dezvoltare a software-ului, de la proiectarea BD la interacțiunea modulelor de cod.

Fiecare tehnică de modelare de mai sus dă o vedere diferită, statică sau dinamică, a unei aplicații. *Colecția de vederi* se numeşte *model*. Iată unele din tehnicile de modelare UML: diagrame de clase, sau diagrame statice de structură, care modelează entitățile unui sistem prin clase cu atribute şi comportare. Diagramele de clasă descriu, de asemenea, asocierile dintre clase şi constrângerile asupra acestora. Apoi, alte tehnici: diagrame de obiecte, diagrame de "caz de utilizare", diagrame de stare, diagrame de secvențe, diagrame de activitate, diagrame de colaborare.

Clase UML

Clasa UML modelează componentele (*entitățile*) de interes ale unui sistem. Clasa are *instanțe*, sau *realizări*. Aceste instanțe sunt *obiectele* clasei. Prin conceptul de clasă se descriu *structura* şi *comportarea* obiectelor clasei. Structura conține atributele fiecărui obiect din clasă. Comportarea include operațiunile ce pot fi executate (efectuate) pe (asupra) o instanță specifică de obiect.

Notația grafică pentru clasa UML:

Nume de clasă	«persistent» Proiect
Atribute	+ proiectId : char + location : string = "Paris" + cost : real
Operațiuni	«create» + newProject(); «destroy» + destroyProject(); «standardAccess» + getProjectId(); # setProjectId(string projectId); + getLocation(); # setLocation(string location); + getCost(); # setCost(real cost);

Stereotipuri: şirul de caractere aşezat între ghilimele (« ») în partea de sus este un stereotip. Un stereotip reprezintă un mod de a extinde semantica unui element de modelare, cum este o clasă. În cazul de față, stereotipul «persistent» identifică clasa Proiect ca fiind o *clasă persistentă*. Clasa persistentă este acea clasă pentru care instanțele sale persistă în memorie după ce procesul care a creat aceste instanțe a luat sfârşit.

În modelarea pentru bazele de date, stereotipul «persistent» indică faptul că o clasă trebuie să fie reprezentată (realizată, "mapped") printr-o reprezentare corespunzătoare a bazei de date în timpul

implementării. O clasă fără stereotipul «persistent» reprezintă un obiect tranzitoriu, care nu persistă în memorie. Corespondențe aici: relații EDB (extensional schema) şi IDB(intensional schema).

Atribute

Sintaxa de specificare a atributelor:
[«stereotype»] [visibility] [/] attributeName [multiplicity] : [type] [=initialValue], în care:

- «stereotype» este un stereotip UML, definit de utilizator, pentru a îmbogăţii semantica (înțelesul!) unei definiții de atribut;
- visibility foloseşte simbolurile:

+ pentru vizibilitate publică (orice clasă poate avea acces la atribut);
# pentru vizibilitate protejată (numai clasa respectivă şi succesorii săi pot avea acces la atribut);
- pentru vizibilitate privată (numai clasa respectivă poate avea acces la atribut);
/ pentru a reprezenta un atribut „derivat";

- attributeName reprezintă numele atributului;
- multiplicity indică dacă atributul este multi-valoare;
- type este un nume de clasă, sau un tip de date, care defineşte tipul valorii ce se memorează în atribut
- initialValue indică o valoare "lipsă" ("default") care trebuie atribuită atributului.

Operațiuni

Sintaxa "prin lipsă" ("default") pentru specificarea semnăturii unei metode este:
[«stereotype»] [visibility] methodName ([parameterList]) : [returnType], în care:

- «stereotype» este un stereotip UML sau definit de utilizator pentru a îmbogăţii semantica (înțelesul) unei operațiuni;
- visibility foloseşte simbolurile:

+ pentru vizibilitate publică (orice clasă poate executa metoda);
# pentru vizibilizate protejată (numai clasa şi subclasele sale pot executa metoda);
- pentru vizibilitate privată (numai clasa poate executa metoda);

- methodName reprezintă numele metodei;
- parameterList este o listă de atribute, sau de perechi de tipuri, separate prin virgule, pentru a indica parametrii formali ai metodei;
- returnType este numele clasei sau tipul de date care indică tipul valorii returnate (întoarse) de o funcție.

În cazul nostru:

- stereotipul «create» este un stereotip UML care identifică o metodă de constituire a instanțelor de obiecte noi ale clasei;
- stereotipul «destroy» este un stereotip UML care indică metoda de "distrugere" a unei clase, pentru a elimina instanțele de obiect;
- stereotipul «standardAccess» este un exemplu de stereotip definit de utilizator, care indică operațiile de atribuire şi de extragere a valorilor de atribut ale unui obiect.

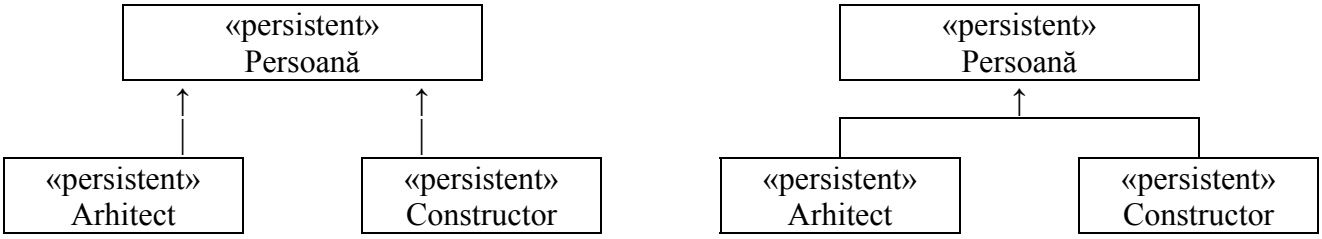
Generalizare. Specializare

În diagramele de clase UML se pot folosi operațiile de *generalizare* şi *specializare* pentru a organiza clasele în ierarhii, de fapt ca în modelul EEA. Notațiile folosite în UML pentru generalizare şi specializare sunt diferite față de cele din EEA, dar sensul este acelaşi. Ca şi în modelul EEA, mulțimea instanțelor pentru fiecare clasă trebuie să respecte restricția ISA, prin care o *instanță a unei subclase* este întotdeauna o *instanță a superclasei sale*.

Restricțiile de specializare disjunctivă şi de specializare totală din EEA sunt valabile şi în UML pentru ierarhiile de clase. De asemenea, diagramele UML acceptă subclase definite prin atribute, deşi într-un mod mai restrâns. Spre deosebire de modelul EEA, diagramele UML admit conceptul de clase abstracte, din programarea OO. Ne amintim că acest tip de clase nu poate fi "instanțiat" în mod direct.

Formarea ierarhiilor de clase în UML

Notații specifice pentru descrierea ierarhiilor claselor în UML:



(a) cu linii separate de subclasă (b) Cu linii de subsclasă arborescente

Între (a) şi (b) nu este diferență semantică.

Notația de mai sus corespunde situației de *apartenență disjunctă* a claselor (apartenența la o superclasă a două subclase disjuncte). Aceasta este o formă de specializare a claselor. Pe lângă reprezentarea de mai sus, de asemenea, în mod explicit, prin intermediul (cu ajutorul) *restricțiilor de specializare*. Acestea sunt construcții (cuvinte, termeni) cuprinse între acolade ("braces") şi asociate cu legătura de specializare între o subclasă şi o superclasă (mai sus se poate asocia {disjunct}).

Restricții de specializare:

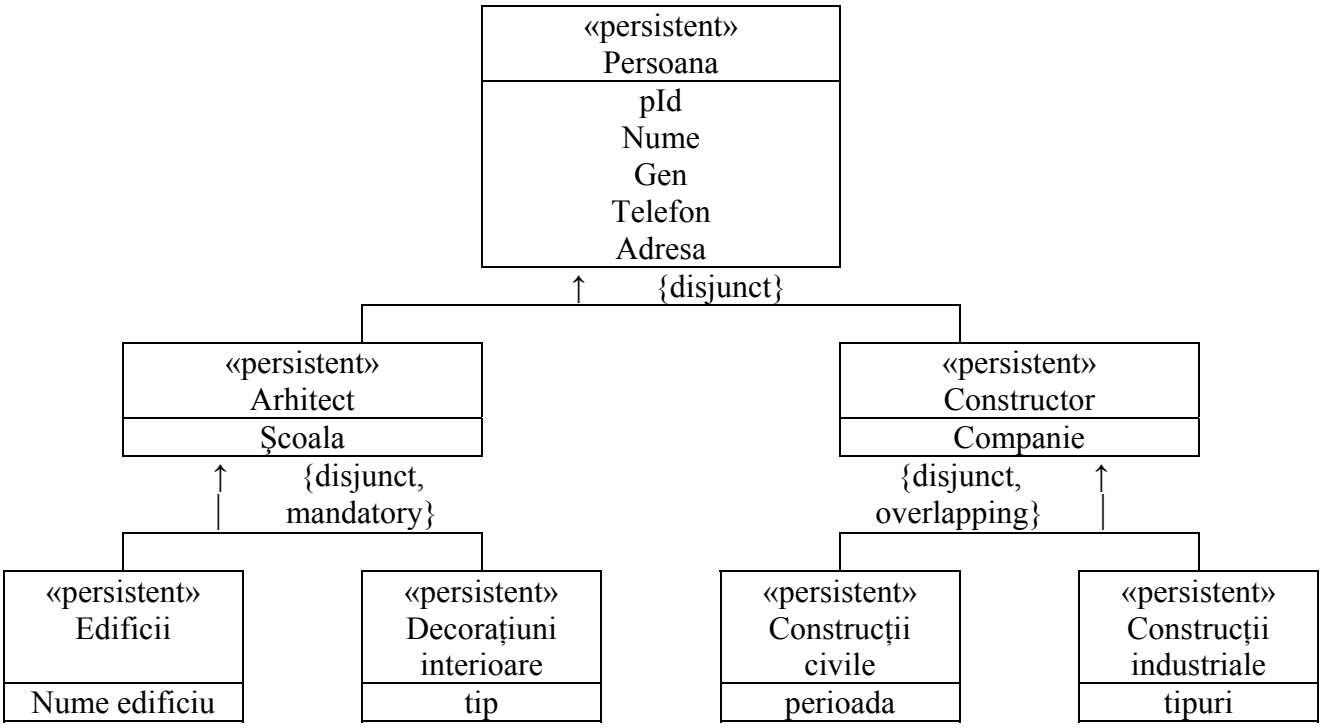
Tip de restricție	Descriere
disjunct	Indică subclase disjuncte
overlapping	Indică subclase care se suprapun
mandatory	Indică participare obligatorie în subclase
complete	Indică faptul că în ierarhie au fost specificate toate subclasele posibile
incomplete	Indică faptul că *nu* au fost specificate toate subclasele posibile

Restricțiile de mai sus se referă la modul şi gradul de participare şi de apartenență a subclaselor în ierarhia de clase. În timp ce *mandatory* este o restricție definită de utilizator, toate celălalte restricții sunt definite de UML.

Note. Restricția *disjunct* indică faptul că intersecția unei mulțimi de subclase este mulțimea vidă. Adică, o instanță a superclasei poate fi o instanță numai a uneia din subclase. *Overlapping* indică faptul că o instanță a superclasei poate fi o instanță a mai multe subclase. *Mandatory* indică că fiecare instanță a superclasei trebuie să fie o instanță a cel puțin uneia din subclasele sale. *Complete* şi *Incomplete* arată că toate subclasele au fost indicate (ie sunt prezente!) în ierarhia de clase.

Ne reamintim că atunci când ne-am ocupat de formarea ierarhiilor de clase în modelul EEA, am folosit exemplul (Persoana) $\in$ (Arhitect, Constructor) $\in$ (Edificii, Decorațiuni Interioare, Construcții Civile, Construcții Industriale), unde (*Persoana*) era în ierarhie *clasa rădăcină*, sau de

bază. Tot în acel context au fost definite tipuri de restricții de specializare pentru modelul EEA (de fapt, care sunt aceleaşi ca în UML). Să reprezentăm acum ierarhia de clase Persoana în UML:



Observăm aici că restricția *mandatory* (de obligativitate) corespunde restricției de *completitudine* din modelul EEA, conducând la o *specializare totală*. Pe de altă parte, absența restricției mandatory corespunde unei *specializări parțiale*.

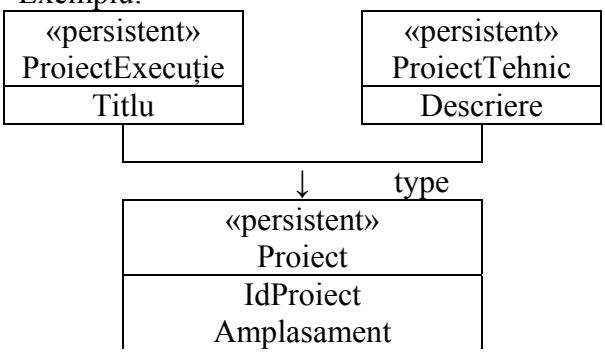
Discriminatoare

În contextul specializării, atât în EEA cât şi în UML, funcționează apartenența unor instanțe la subclase pe baza valorii unui atribut de nivel de superclasă. Acest atribut, care determină apartenența la o subclasă, se numeşte *discriminator*.

Observăm că în UML folosirea unui discriminator implică întotdeauna *specializările* de tip *disjunct* şi *mandatory*, respectiv aceste tipuri de apartenență la nivel de subclasă. Iar o subclasă trebuie să existe pentru toate valorile discriminatorului.

Discriminatorul este specificat ca o etichetă pe legătura de specializare şi este considerat ca fiind un pseudoatribut al superclasei. Pseudoatributul se comportă ca un atribut al unei clase obişnuite, însă domeniul său este mulțimea numelor de subclasă. Remarcăm că un pseudoatribut nu apare în zona de atribute de definiție a clasei.

Exemplu:



O clasă abstractă este indicată cu numele său în italic sau cu {abstract} sub numele clasei

Mai sus, discriminatorul este pseudoatributul type. Valori posibile pentru type sunt ProiectExecuţie şi ProiectTehnic. Discriminatorul implică specializare de tip mandatory la nivel de subclase. Ca urmare, clasa Proiect de mai sus este o *clasă abstractă* (de obicei, în italice). O clasă abstractă nu poate fi instanţiată direct (adică instanţele sale nu se pot crea direct). În cazul nostru, un obiect al clasei Proiect poate fi creat numai ca o instanţă a clasei ProiectExecuţie sau a clasei ProiectTehnic. Prin restricţia ISA, obiectul va fi automat şi o instanţă a clasei Proiect.

Într-o ierarhie de clase pot fi mai multe discriminatoare. În acest caz, o instanţă a unei superclase trebuie specializată pentru fiecare discriminator, iar subclasele fiecărui discriminator trebuie să fie clase abstracte. Avem moştenire multiplă în acest caz (multiple instance).

Clase abstracte şi clase concrete

O clasă abstractă primeşte instanţe în mod *indirect*, prin instanţierea subclaselor sale. Drept rezultat, o *clasă abstractă* are întotdeauna *cel puţin un nivel de subclase*. Clasele abtracte oferă o manieră mai directă, orientată obiect, pentru a specifica participarea totală, aşa cum aceasta este definită prin *restricţia de completitudine a modelului EEA*. Fiecare instanţă a unei superclase abstracte este întotdeauna o instanţă a cel puţin unei subclase, deoarece obiectele pot fi create numai la nivel de subclasă.

Clasele concrete sunt cele care pot fi *instanţiate direct*. În ierarhia din diagrama UML pentru Persoana, toate clasele sunt concrete. În ultimul exemplu, doar clasele ProiectExecuţie şi ProiectTehnic sunt concrete.

În tratarea diagramelor de clase se întâlnesc *operaţiuni abstracte* şi *operaţiuni concrete*. O operaţiune abstractă nu are implementare. Ea este întotdeauna specificată într-o clasă abstractă, numele său fiind dat în italice sau cu {abstract} după semnătura metodei. Implementarea este realizată la nivelul unei *clase concrete*, care *moşteneşte această operaţiune*. O operaţiune concretă este aceea care are o implementare. Clasele abstracte pot avea operaţiuni concrete.

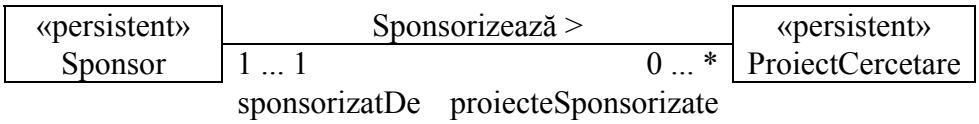
Asocieri

Ca şi în modelul EEA, legăturile de diverse tipuri între mulţimile entitate (1:1, 1:N, M:N) sunt denumite în UML *asocieri* (associations), *nu "relationship"*, ca în modelul de bază EA. Deci mai avem o *problemă de terminologie*, deoarece atunci când am introdus modelul EA am tradus termenul "relationship" prin asociere. Diagramele de clase UML acceptă "legăturile" de tip 1:1, 1:N, M:N între clase, fiind folosit *termenul de asociere*.

Asocieri de bază în UML

Pentru a indica o asociere între două clase, se trasează o *linie*. Cu aceste linii se pot asocia: *nume de asocieri*, *nume de roluri*, *multiplicităţi*, în vederea *îmbogăţirii conţinutului semantic* al descrierii asocierii.

Exemplu:



Se observă evoluţia metodologiei de proiectare conceptuală – inclusiv în simbolurile utilizate în modelare.

Terminologie:

Numele asocierii	sponsorizează	O săgeată indică direcția citirii legăturii („relationship”)
Nume de roluri	sponsorizatDe	Un ProiectCercetare este sponsorizatDe un sponsor
	proiecteSponsorizate	Un sponsor are mai multe proiecteSponsorizate
Multiplicități	1 ... 1	Un proiectCercetare trebuie sponsorizatDe numai un singur sponsor
	0 ... *	Un sponsor poate să nu sponsorizeze nici unul, dar poate sponsoriza mai multe ProiecteCercetare

Ne reamintim că ideea de multiplicitate am întâlnit-o anterior, atunci când am folosit termenii de cardinalitate şi opționalitate. Termenul multiplicitate este folosit în etapa actuală a modelării BDOR. Termenii de cardinalitate şi opționalitate erau folosiți într-o fază anterioară (10-12 ani!), cea în care se afirma modelul EEA, iar UML nici nu apăruse.

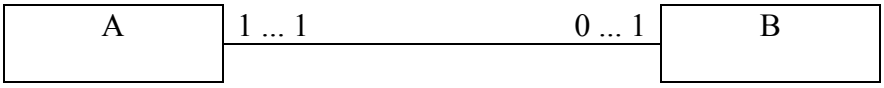
Notația pentru *multiplicitate* în cazul *asocierilor UML* este prezentată mai jos:

Notație min ... max (se referă la cel puțin min obiecte şi la cel mult max obiecte)	0 ... *	Se referă la zero sau mai multe obiecte
	0 ... 1	Se referă la niciun obiect sau la cel mult un obiect
	1 ... *	Se referă la cel puțin un obiect
	1 ... 1	Se referă la exact un obiect
	3 ... 5	Se referă la cel puțin trei obiecte şi la cel mult cinci obiecte
Notație prescurtată (shorthand)	1	La fel ca 1 ... 1
	*	La fel ca 0 ... *

Multiplicitățile în UML sunt similare cu perechile (min, max) din modelul EEA, dar sunt notate cu *min ... max*. În UML, atunci când descriem cu câte obiecte din clasa B poate fi asociat un obiect din clasa A, pereche min ... max este amplasată, pe linia asocierii, lângă clasa B. Acest aranjament este diferit față de cel al perechii (min, max) în cazul modelului EEA; în acest ultim caz, perechea (min, max) este asociată cu linia care uneşte clasa cu *rombul legăturii*, indicând de câte ori – minimum şi maximum – un obiect al clasei participă la legătură (“relationship”). Valorile *min* şi *max* au însă aceeaşi semnificație cu privire la restricțiile de cardinalitate şi de participare din modelul EEA. De exemplu, un min pentru zero indică întotdeauna o *participare parțială* în asociere, pe când un min cu o valoare mai mare sau egală cu unu indică o *participare totală*.

Este interesant să examinăm comparativ notațiile UML şi EAE, aşa cum sunt arătate mai jos. Am ales cazul asocierilor 1:1.

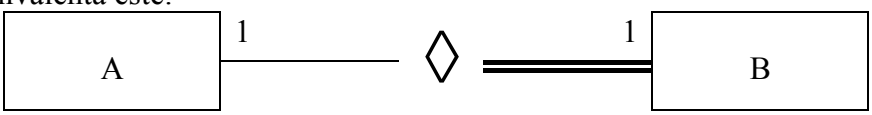
Notația UML



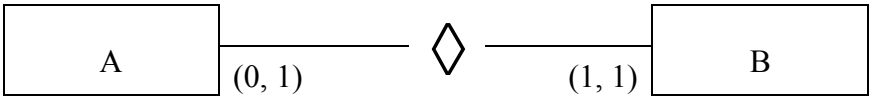
Clasa A (un obiect din clasa A!) este asociată cu 0 ... 1 obiecte din clasa B (participare parțială)
Clasa B (un obiect din clasa B!) este asociată cu 1 ... 1 obiecte din clasa A (participare totală)

Notația EAE echivalentă este:

EAE



EAE cu (min, max)



În figura de mai sus, în asocierea dintre clasele A şi B, obiectele din A au o participare parțială în asociere, indicată prin valoarea min de 0 în perechea (min ... max) asociată cu B. În consecință, este

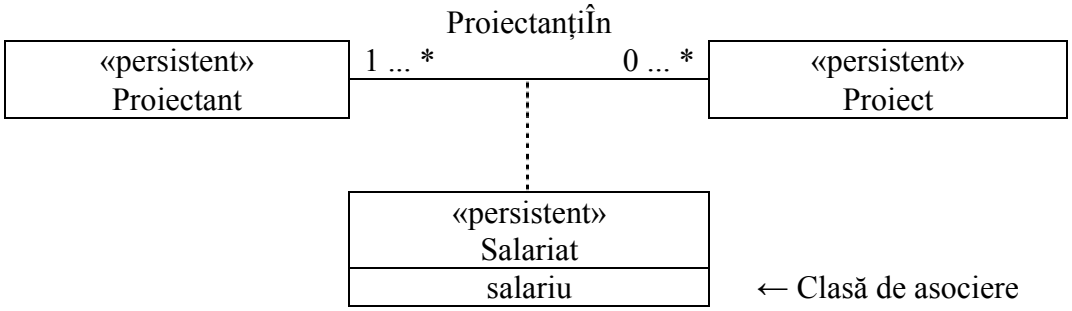
posibil pentru o instanță din A să fie creată fără a se stabili o asociere cu un obiect din B. Pe de altă parte, obiectele din B au o participare totală, indicată de valoare min de 1 în perechea (min ... max) asociată cu clasa A. Notăm că fiecare obiect din B trebuie să fie asociat cu exact un obiect din A.

În dezvoltarea diagramelor de clase UML pentru proiectarea conceptuală a unei BD (obiectual – relațională!), folosirea numelor de asocieri, a numelor de roluri şi a multiplicităților este opțională. În fazele de început ale procesului de proiectare a unei baze de date, este acceptabil ca între două clase să fie trasată numai o linie, pentru a marca faptul că există o asociere. Totuşi, detaliile asocierii urmează a fi precizate (rafinate) în continuare, în procesul de proiectare. Cel puțin, trebuie specificate multiplicitățile înainte ca o diagramă de clase UML să fie aplicată (mapped) pe un model specific de implementare (deci, transformată într-o implementare specifică).

Clase de asocieri şi asocieri materializate

Ne reamintim că în modelul EAE, asocierile (numite în acel model "relationships") pot avea atribute. De exemplu, în cadrul unei asocieri M:N între mulțimile entitate Proiectant şi Proiect, un proiectant primeşte un salariu diferit pentru fiecare proiect la realizarea căruia participă. Desigur, valoarea salariului nu poate fi un atribut al entităților Proiectant, deoarece un proiectant poate avea un salariu diferit pentru fiecare proiect la care lucrează. De asemenea, nici entitățile Proiect nu pot avea valoarea salariului ca atribut, întrucât un proiect poate folosi câțiva proiectanți, care în general au salarii diferite. Modalitatea corectă de a modela această situație este ataşarea unui atribut a asocierea mulțimilor entitate Proiectant şi Proiect.

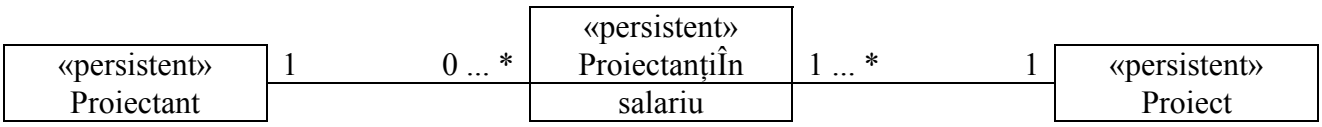
În diagramele de clase UML, atributele asocierilor pot fi modelate cu ajutorul *claselor de asocieri*. Un exemplu – în continuare, pentru asocierea M:N cu numele ProiectanțiÎn, între mulțimile entitate Proiectant şi Proiect. În UML, la asocierea menționată, se ataşează o clasă de asociere, prin intermediul unei linii întrerupte, aşa cum se arată în figura de mai jos.



O *clasă de asociere* are o *zonă de nume*, pentru numele clasei, în cazul nostru Salariat. De asemenea, o clasă de asociere poate avea o *zonă de atribute* (în cazul nostru numai atributul salariu, care indică ce salariu primeşte un proiectant pentru fiecare proiect la care participă), precum şi o *zonă de operațiuni*. De notat că o clasă de asociere poate participa în asocieri cu alte clase.

O *altă notație* legată de folosirea claselor de asociere este cea numită *asocierea materializată (sau reificată)*. Într-o asociere materializată, asocierea se modelează ca o clasă, procesul de transformare a unei asocieri în clasă fiind numit materializare, sau reificare. În această abordare, ceva care în aplicația studiată nu este înțeles de obicei ca un obiect – ca de exemplu, o asociere între două clase – , este modelat printr-o clasă.

Să vedem cum se aplică procedeul de materializare (reificare) în cazul exemplului anterior. Evident, asocierea între clasele Proiectant şi Proiect va fi transformată acum într-o clasă de sine stătătoare, cu numele ProiectanțiÎn (am folosit chiar numele asocierii), aşa cum se vede mai jos:



Aşa cum se vede, clasa Salariat a încetat să fie o clasă de asociere şi a devenit o clasă UML propriu-zisă, cu numele ProiectanțiÎn. Asocierea binară M:N între clasele Proiectant şi Proiect a fost transformată în două legături (“relationships”) binare separate de tip 1:N – una din ele leagă Proiectant cu ProiectanțiÎn, iar cealaltă leagă Proiect cu ProiectanțiÎn.

În noua abordare (reprezentare), fiecare instanță a clasei ProiectanțiÎn este un obiect, care reprezintă o legătură (asociere) între un proiectant şi un proiect (de exemplu, pe partea 1 a fiecărei asocieri 1:N). De asemenea, deoarece un proiectant ar putea participa (lucra) la mai multe proiecte, partea N a asocierii 1:N între Proiectant şi ProiectanțiÎn este descrisă ca fiind o multiplicitate 0...*. Aşadar, prin procesul de materializare (reificare) asocierea M:N care conține o clasă de asociere a fost reprezentată indirect prin două asocieri 1:N, iar asocierea a fost reprezentată explicit printr-o clasă de sine stătătoare.