Universitatea "Politehnica"
Timişoara
Facultatea de Automatizări şi Calculatoare

Baze de Date Orientate Obiect pentru Sisteme de Fabricaţie

TEZĂ DE DOCTORAT

Timişoara
2003

CUPRINS

Introducere

Istoria cercetării în domeniul bazelor de date, este caracterizată de o excepțională productivitate şi de un uimitor impact economic. Domeniu de cercetare ştiințifică fundamentală de cel mult 30 de ani, investigarea bazelor de date a alimentat o industrie a serviciilor de informații estimată la miliarde de dolari pe an, numai în SUA. Rezultatele cercetării în domeniul bazelor de date constitue temelia progreselor fundamentale din sistemele de comunicații, transport, logistică, gestiune financiară, sisteme bazate pe cunoştințe şi accesibilitate la literatura ştiințifică, găzduind totodată alte aplicații civile şi din domeniul apărării. De asemenea, acestea constituie baza progreselor considerabile în domeniul ştiințelor fundamentale, de la calculatoare la biologie. Bazele de date formează acum, cadrul sistemelor informaționale şi au modificat fundamental modul de operare al organizațiilor. Tehnologia bazelor de date reprezintă un domeniu incitant şi încă de la apariția sa, a fost un catalizator pentru multe realizări semnificative în ingineria programării. Acum, bazele de date, fac parte din viața noastră de zi cu zi în aşa măsură, încât adeseori nu suntem conştienți că le utilizăm.

În acest context, paradigma orientării spre obiecte, poate fi un instrument propice pentru rezolvarea tendințelor în diverse domenii. Acesta se datorează faptului că, un obiect furnizează o abordare mult mai bună în vederea creerii bazelor sistemelor, decât abordarea structurată. Obiectele software pot constitui fundamentul unei puternice viziuni asupra procesării informației. Într-un context mai larg obiectul este folosit pentru a face referință la o structură, un proces sau resursă, împreună cu procedurile de acces: procesoare, baze de date, interfețe grafice, toate orientate spre obiecte. Obiectul software şi obiectul din lumea reală sunt entități total diferite, obiectul software fiind de fapt interpretarea particulară a celui real.

Metodologia orientată pe obiecte este instrumentul care realizează proiectarea bazelor de date, realizând un sistem software complex, care cuprinde atât datele cât şi metodele ataşate obiectelor. Tehnologia obiectuală facilitează urmărirea evoluției unui produs pe toată perioada sa de existență, mult mai exact decât tehnologia structurată.

Utilizarea rațională a tehnologiei informației în sistemele de fabricație, este una dintre orientările strategice în vederea atingerii obiectivelor de integrare, de automatizare şi de flexibilitate a producției, care ar trebui să le permită acestora să devină mai competitive. Aceste obiective determină redefinirea şi adesea simplificarea proceselor întreprinderii (procese de decizie,

procese de fabricație, procese administrative), precum și gestionarea continuă a nevoilor de schimbare a acestor procese, în funcție de evoluția piesei sau a mediului înconjurător întreprinderii.

Automatizarea flexibilă a proceselor tehnologice de prelucrare reprezintă în prezent, pe plan mondial, pilonul central al procesului de obținere a sistemelor integrate de producție, caracterizate prin asocierea unor echipamente și utilaje complexe, cu sisteme informatice de înaltă performanță, asamblând, într-o viziune ierarhică unitară, funcțiile de control, manipulare, transport, depozitare, alături de funcția de prelucrare.

Fiind sisteme înalt automatizate, indiferent de natura proceselor comandate, sistemele de fabricație sunt conduse de calculatoare electronice. În scopul îmbunătățirii modului de funcționare, pentru a le mări gradul de flexibilitate și în același timp productivitatea, aceste sisteme necesită structuri software tot mai evoluate.

Un sistem de fabricație conține o bază de date de tip CAD (Computer Aided Design), aferentă proiectării asistate de calculator, o bază de date de tip CAM (Computer Aided Manufacturing), care stochează date necesare în procesul de fabricare a produselor și baze de date de tip OIS (Office Information System), care deservesc sisteme informaționale de birou.

Baza de date CAD, stochează date privind proiectarea mecanică sau electrică. Datele de proiectare sunt caracterizate printr-un număr mare de tipuri, fiecare cu un număr mic de apariții. Proiectele pot fi laborioase, ele pot conține un număr foarte mare de părți. Proiectarea nu este statică, ea evoluează în timp, reactualizările se propagă în profunzime. În activitatea de proiectare pot fi implicate foarte multe persoane.

Baza de date CAM, stochează date similare cu cele ale unui sistem CAD. Pe lângă acestea, stochează date care se referă la producția discretă și la producția continuă. În acest caz, sistemul trebuie să înmgazineze un volum mare de date, cu o structură complexă. De asemenea, el trebuie să navigheze rapid printre date și să răspundă la modificări.

O bază de date de tip OIS stochează date privind controlul pe calculator al datelor de afaceri, documente de natură economică, documente referitoare la personal. Sistemele moderne manipulează documente multimedia, adică date cu o structură mult mai complexă decât datele cu structură alfanumerică.

Toate aceste probleme complexe care apar, au rezolvare în *sistemul de baze de date orientate obiect* (OODBMS) sau *sisteme de baze de date relațional obiectuale (RODBMS).*

În această teză principalul obiectiv este acela de a cerceta modul în care bazele de date pur obiectuale și cele relațional obiectuale, pot fi folosite în cadrul unui sistem de fabricație, în particular Sistemul de Fabricație al Sculelor. Sunt tratate atât aspectele teoretice, cât și cele practice. Din punct de vedere teoretic, sunt dezvoltate noțiuni puțin tratate în literatura de specialitate, ca de exemplu clase derivate și scheme externe. Din punct de vedere practic, este

realizată implementarea originală a bazei de date pentru sistemul de fabricație, atât într-un mediu pur obiectual (Jasmine), cât şi într-un mediu relațional obiectual (Visual Fox). Contribuțiile pe care le aduce teza, pot fi catalogate pe mai multe planuri. Din punct de vedere teoretic, sunt definite concepte noi pentru clasele derivate şi schemele externe, sunt cercetate modalități noi de implementare a claselor derivate, algoritmi noi de creare a schemelor externe, modalități de îmbogățire a standardului ODMG 3.0, clasificare originală a sistemului de fabricație. Cercetarea efectuată pentru implementarea obiectuală şi relațional obiectuală, a scos în evidență particularități privind realizarea bazei de date în cele două medii, atât din punct de vedere al funcționalității cât şi din punct de vedere al programării orientate obiect, al modului în care sunt tratate obiectele ce aparțin claselor sistemului de fabricație. Noțiunile teoretice, clasă derivată şi schemă externă au fost implementate practic în aplicația realizată pentru sistemul de fabricație, în ambele medii obiectuale. Cercetarea efectuată pentru implementarea relațional obiectuală, a determinat abordarea originală, pur obiectuală, prin faptul că au fost realizate clase pentru generarea suportului de memorare, care transformă relația în obiect.

În capitolul 1 se face o analiză a fundamentelor obiectuale: obiect complex, identitatea obiectului, încapsulare, tipuri şi clase, moştenire, redefinire statică, polimorfism, scoțându-se în relief principalele lor caracteristici şi modul de folosire în cazul sistemelor de fabricație.

În capitolul 2 se face o cercetare asupra principalelor caracteristici ale bazelor de date obiectuale: concurență, tranzacții, refacere, interogare, versionare, persistență, securitate, arhivare. Sunt prezentate principalele tipuri de baze de date obiectuale, din punct de vedere al modului de tratare al obiectelor, standardizări existente ale OODBMS-urilor, precum şi arhitecturi ale acestora.

Proiectarea bazei de date, este un aspect important, prezentat în capitolul 3. Pe lângă analiza etapelor care trebuie parcurse la proiectarea bazei de date în general, se face o cercetare amănunțită a modului cum acestea trebuie aplicate în cazul particular al bazei de date obiectuale. În acest capitol sunt scoase în evidență particularitățile proiectării bazei de date în mediul obiectual.

Schemele externe şi clasele derivate, sunt noțiuni tratate relativ puțin în literatura de specialitate. În capitolul 4 sunt analizate principalele metodologii de realizare a schemelor externe (obiectuale), a modului de definire a claselor derivate, folosind diverse tipuri de relații. Pornind de la o arhitectură specifică mediilor relaționale, se realizează adaptabilitatea acesteia la mediul obiectual. De fapt, cercetarea prin care se apropie relaționalul de obiectual, este una dintre activitățile de bază ale acestei teze. Este definită noțiunea de clasă transformabilă şi clasă netransformabilă, în ideea de a obține scheme externe cât mai flexibile şi de a mări puterea de informare a acestora. Sunt realizate noi definiții privind ordinea de modificare a claselor transformabile în ierarhia de clase, moştenirea între clase transformabile şi netransformabile. Modul în care

sunt integrate clasele derivate în schema obiectuală, este un alt aspect important dezvoltat în acest capitol.

Pe baza conceptelor studiate şi dezvoltate în capitolul 4, în capitolul 5 sunt prezentate metodologii noi, originale, privind definirea claselor derivate şi a schemelor externe. Totodată în acest capitol sunt prezentaţi doi noi algoritmi de realizare a schemelor externe, unul în cazul în care schemele externe conţin numai clase transformabile şi unul în cazul în care schemele externe conţin atât clase transformabile cât şi clase netransformabile. În acest capitol sunt dezvoltate semanticile de generare a claselor derivate. Se arată posibilitatea de a folosi semantica obiect generat, pentru a obţine o clasă derivată, într-o schemă obiectuală a unui sistem de fabricaţie, în contradicţie cu informaţiile existente în literatura de specialitate. Este interpretată în mod original informaţia existentă într-o clasă derivată, acesta putând conţine proprietăţi nederivabile. În acest fel clasa derivată poate fi parţial derivabilă.

Capitolul 6 conţine informaţii despre sistemele de fabricaţie, în special despre Sistemul de Fabricaţie al Sculelor. Se realizează o clasificare originală a acestuia, pe baza căreia se vor dezvolta bazele de date pur obiectuale şi relaţional obiectuale.

Capitolele 7 şi 8, prezintă implementarea în mod original a bazei de date pur obiectuale în software-ul Jasmine şi cea relaţional obiectuală în software-ul Visual Fox. Se face un studiu comparativ privind modurile de implementare, particularităţile pe care le aduce implementarea intr-un mediu pur obiectual. Implementarea în mediul relaţional obiectual s-a făcut pornind de la faptul că în momentul de faţă acest tip de baze de date este cel mai răspândit pe plan mondial. Se face o cercetare amănunţită a particularităţilor pe care le impune o astfel de implementare. Prin crearea prin cod program a unor clase de obiecte care au posibilitatea de a memora datele, s-a realizat o apropiere semnificativă de modul de tratare pur obiectual al informaţiilor. În proiectarea şi realizarea bazei de date relaţional obiectuale ca şi în cazul celei pur obiectuale, s-au aplicat noţiunile teoretice noi, prezentate în capitolele tezei. S-a dat o interpretare mai largă noţiunii de clasă derivată, aceasta fiind mai mult decât o simplă clasă vedere. Prin studiul compartiv între implementarea pur obiectuală şi relaţional obiectuală, a sistemului de fabricaţie, s-au obţinut noi perspective privind realizarea bazelor de date în mediul industrial.

Contribuţiile şi perspectivele acestei teze sunt prezentate în capitolul 9.

Proiectarea şi implementarea bazelor de date în mediile CAD, CAM, oferă posibilităţi multiple de realizare. Această teză demonstrează faptul că varianta obiectuală a bazei de date, este alegerea optimă în cazul sistemelor de fabricaţie, prin multitudinea soluţiilor pe care le oferă.

1

Fundamente Obiectuale

La sfârşitul anilor 80 şi începutul anilor 90, complexitatea crescândă a informaţiilor a făcut ca o serie de specialişti în baze de date să gândească un nou model mai apropiat de noile cerinţe, reorganizând cunoştiinţele existente. Acest efort de reorganizare a condus la elaborarea unor tehnologii despre bazele de date bazate pe concepte orientate obiect.

Fenomenul orientării spre obiecte este un lucru vechi. Filozofii antici Platon şi Aristotel şi cei moderni, Kant, au încercat explicarea sensului existenţei şi caracteristicile esenţiale ale obiectelor. Un obiect modelează o entitate din lumea reală sau imaginară. Astele tot ce putem gândi, vedea, atinge sau clasifica întrun anume mod reprezintă obiecte

Programarea este un proces complex care implică ştiinţă, imaginaţie şi tehnică de calcul. Datorită dificultăţilor pe care le ridică acest proces s-a simţit tot mai acut necesitatea unei sistematizări a procedeelor folosite în elaborarea programelor. Tendinţa curentă de reingeneering şi sistemele client/server au determinat în măsură tot mai mare folosirea paradigmei orientată obiect în vederea găsirii unui instrument de dezvoltare al aplicaţiilor

Conform [Atkin 89] definiţia orientării obiect este următoarea:

> O mulţime de principii de proiectare şi dezvoltare bazate pe conceptul de structură autonomă
> Fiecare structură autonomă, reprezintă o entitate a lumii reale, care are abilitatea de interacţiona cu ea însăşi sau cu alte obiecte

Obiectele software constitue blocurile structurale ale unei noi şi puternice viziuni asupra procesării informaţiei. Ele sunt utilizate pentru a referi o structură, resursă sau proces împreună cu procedurile de acces: procesoare, baze de date, interfeţe utilizator.

Caracteristicile importante ale unui obiect sunt arătate în [Booch 94]:

- *Starea* sa;
- *Comportamentul* său;
- *Identitatea* sa.

Starea obiectului sau memoria sa reprezintă mulțimea datelor care conțin informații despre acesta.

Identitatea obiectului este un cod care individualizează unic obiectul.

Comportamentul obiectului este definit de modul în care acesta interacționează cu alte obiecte, precum și de modul în care reacționează la diverși factori externi.

Un sistem orientat obiect trebuie să îndeplinească o serie de caracteristici. Aceste caracteristici pot fi împărțite în trei grupe:

- *Obligatorii*, pe care sistemul trebuie să le satisfacă în ordine pentru a fi orientat obiect. Acestea sunt complexitatea obiectelor, identitatea obiectelor, încapsularea, tipuri de clase, moștenirea, rescrierea combinată cu construcția ulterioară, extensibilitatea, persistența, managementul memorării datelor, concurența și facilitățile de interogare;
- *Opționale*, care pot fi folosite pentru a face sistemul mai bun, dar care nu sunt obligatorii. Acestea sunt moștenirea multiplă, deducerea, distribuția;
- *Deschise*, punctele unde proiectantul poate folosi mai multe opțiuni. Acestea sunt: paradigmele programării, reprezentările sistem, uniformitatea și tipurile de sistem.

În general, problema prin care un anumit sistem este definit orientat obiect, poate face subiectul unor îndelungate dezbateri. Problema poate fi analizată din trei puncte de vedere:

- Lipsa unui model comun de date;
- Lipsa unor fundamente formale;
- Puternică activitate de experimentare.

Pentru bazele de date relaționale există un puternic fundament teoretic emis de către Codd. La un moment dat acestea erau cele mai răspândite baze de date. La sfârșitul anilor 80 și începutul anilor 90 au apărut primele fundamente referitoare la noile baze de date, cele orientate obiect. La începutul anilor 90 au cunoscut o puternică fundamentare teoretică, astfel încât se prevedea că sfârșitul de mileniu va fi în domeniul bazelor de date, obiectual. Practica a arătat că acest lucru nu a fost în totalitate adevărat, la momentul actual sistemele relațional-obiectuale fiind cele mai răspândite. Sistemele orientate obiect își găsesc o largă aplicabilitate în sistemele CAD, CASE. În aceste sisteme se vehiculează un volum mare de date, acestea sunt în general date complexe, astfel încât sistemul trebuie să se caracterizeze printr-o înaltă performanță dictată de către sistemele interactive. Una dintre lucrările fundamentale, care definește conceptele orientate obiect, este [Atkin 89]. Această lucrare este prima care definește cerințele necesare unei baze de date, să fie orientată obiect. "Manifesto" definește treisprezece cerințe obligatorii și altele opționale.

1.1 Obiecte complexe

Obiectele complexe sunt construite din obiectele simple, aplicând constructori de obiecte. Obiectele simple sunt obiecte, ca de exemplu: întregi, şiruri de caractere, variabile booleene. Ca şi constructori de obiecte se pot considera: tuple, mulțimi, liste, tablouri (matrici). Setul minim de constructori pe care-l poate avea un sistem este format din: mulțimi, tuple şi liste. Tuplele sunt necesare pentru că ele sunt o cale naturală de reprezentare a proprietăților unei entități, listele şi matricile sunt importante pentru că ele reprezintă obiecte din mediul exterior şi sunt folosite în diverse aplicații ştiințifice.

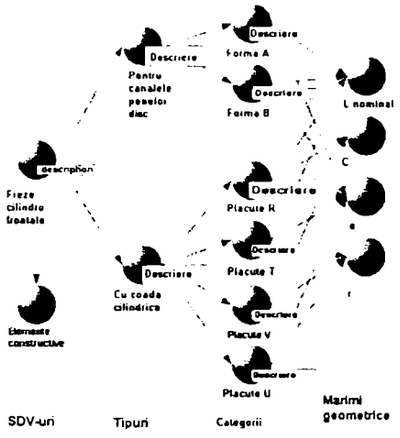


Fig 1.1 Model de obiect

Constructorii de obiecte pot fi ortogonali, adică fiecare constructor poate fi aplicat la un obiect. Constructorii dintr-un model relațional nu sunt ortogonali pentru că ei se aplică numai tuplelor, care la rândul lor se aplică doar la valorile atomice. Obiectele complexe presupun că operatorii potriviți trebuie să se repartizeze uniform la toate obiectele. Aceasta înseamnă că operațiile care se efectuează asupra unui obiect complex se repartizează uniform asupra tuturor componentelor sale.

Unele modele de obiecte complexe se bazează pe valoare, adică pe faptul că unicitatea unui obiect depinde doar de starea sa, care constă în valori elementare întregi ale variabilelor de instanță. Un astfel de model este reprezentat în figura 1.1.

Obiectele complexe pot fi referite cu ajutorul identității lor, un alt concept fundamental al programării orientate obiect. Un astfel de sistem este reprezentat în figura 1.2. Fiecare obiect primeşte un cod de identificare care îl individualizează de celelalte obiecte. Codul de identificare poate fi format dintr-un şir de carectere.

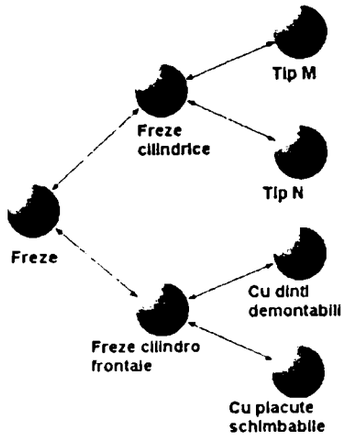


Fig 1.2 Obiecte complexe referite cu ajutorul identității

1.2 Identitatea obiectului

Acest concept, care există de mult timp în limbajele de programare, este relativ recent în domeniul bazelor de date. Ideea conceptului, constă în faptul că, un obiect poate avea o existență independentă de valoarea sa. *Identitatea* obiectului este proprietatea unui obiect care îl deosebeşte de toate celelalte obiecte. Cel mai obişnuit tip de identitate de obiect folosit în sisteme de operare, limbaje de programare sau baze de date, constă în nume al obiectelor, definite de utilizator. Folosind identitatea obiectelor, acestea pot referi alte obiecte. Datorită acestui fapt, există două noțiuni echivalente pentru obiect: două obiecte pot fi identice sau pot fi egale. Testul de identitate verifică dacă două obiecte sunt de fapt unul singur, pe când testul de egalitate verifică dacă conținutul a două obiecte este acelaşi.

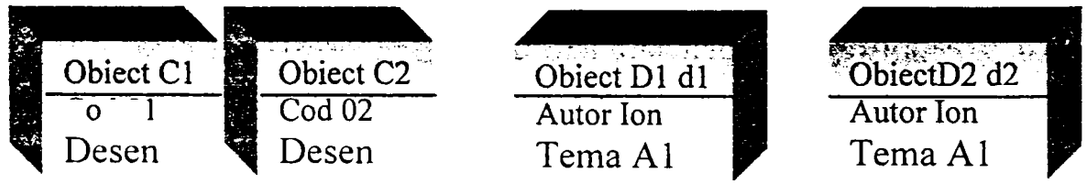


Fig 1.3 Grup de obiecte

Considerăm obiectele C1 şi C2 din clasa "Cuțite" şi obiectele D1 şi D2 din clasa "Desene", figura 1.3. Din figură se observă că obiectele D1 cu adresa d1 şi D2 cu adresa d2 reprezintă acelaşi obiect, având acelaşi conținut. Se poate spune că cele două obiecte D1 şi D2 sunt egale. Pe de altă parte, având adrese diferite d1 şi d2, afirmația C1.Desen = = C2.Desen este falsă, dar afirmația C1.Desen=C2.Desen este adevărată.

Aceasta are două implicații asupra obiectelor: *obiect partajat şi obiect actualizat.* Modelul *obiect partajat* presupune că două obiecte fac referință la aceeaşi componentă. *Obiect actualizat* presupune modificări asupra tuturor obiectelor care partajează o anumită componentă. Identitatea bazată pe model

este o componentă a limbajelor de programare, în sensul în care fiecare obiect are o identitate şi poate fi actualizat (updated).

O altă modalitate de identificare a obiectelor este folosirea unei chei unice sau a unei chei de identificare. Acest mecanism este folosit frecvent în sisteme de administrare a bazelor de date. La utilizarea cheilor de identificare pentru identificarea obiectelor apar trei probleme principale:

- *modificarea cheilor de identificare* nu se poate efectua în procesul de administrare a unei baze de date;
- *neuniformitatea,* constă în faptul că, cheile de identificare se adresează tipurilor diferite de date, astfel ele pot fi şiruri de caractere, numere întregi etc. Un alt lucru care poate apărea este ca atributele folosite drept chei de identificare să se schimbe;
- *legături nenaturale,* constau în faptul că, legăturile făcute după cheile de identificare sunt folosite la căutări, în loc să se găsească metode de căutare mai simple şi mai directe.

În general, fiecare obiect este o instanță a unei clase şi fiecare obiect are o stare care este valoarea variabilelor sale de instanță. Totodată fiecare obiect are o identitate care este stabilită intern şi care este diferită de clasa sau de starea sa. *Identitatea obiectului* este generată, când se crează obiectul şi ea este permanentă, pe când starea sa, se poate modifica în timp. Sistemele orientate pe obiecte, care asigură suport pentru identitatea internă permit, de asemenea, obiectului să sufere modificări structurale (ceea ce înseamnă modificarea clasei), fără a-şi schimba identitatea. Fără identitate sau alte mijloace de referire ale obiectelor independente de starea lor, este imposibil ca acelaşi obiect să fie valoarea variabilei de instanță a mai mult de un obiect.

1.3 Încapsularea

Ideea de încapsulare vine din informatică şi face referință la tipurile de date abstracte, ca de exemplu stiva, asupra căreia se pot face operații de inițializare, introducere, scoatere sau poziționare. Din acest punct de vedere, un obiect are două părți distincte: o parte de implementare şi o parte de interfață. Partea de interfață este partea vizibilă a obiectului şi ea specifică mulțimea de operații care pot fi efectuate asupra obiectului. Partea de implementare este formată din partea de date şi partea procedurală.

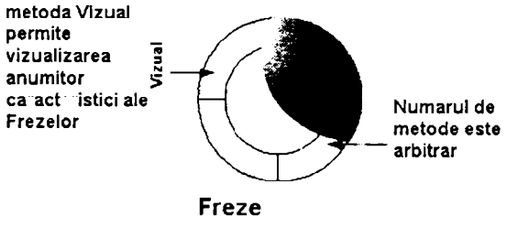


Fig 1.4 Incapsularea obiectului

Partea de date reprezintă starea obiectului iar partea procedurală descrie implementarea fiecărei operații.

Într-un sistem care admite noțiunea de încapsulare, un obiect al bazei de date, identificat printr-un nume, se caracterizează prin operațiile aplicate asupra sa şi mai puțin prin proprietățile sale fizice. Cu alte cuvinte, obiectele bazei de date care permit încapsularea conțin atât date cât şi proceduri. Operațiile care pot fi efectuate asupra obiectului sunt vizibile, pe când datele şi procedurile sunt ascunse în raport cu alte obiecte.

1.4 Tipuri şi clase

Sistemele orientate obiect pot fi împărțite în două mari categorii:

- Cele care suportă noțiunea de tip: C++, Simula, Trllis/Owl, O2;
- Cele care suportă noțiunea de clasă: Smalltalk, Gemstone, Vision, Orion etc;

Într-un sistem orientat obiect, tipul rezumă trăsăturile comune a unei mulțimi de obiecte cu aceleaşi caracteristici. El corespunde noțiunii de dată abstractă şi este format din două părți: interfața şi implementarea. Din punct de vedere al limbajelor de programare, tipul defineşte o structură folosită la controlul corectitudinii unui program în timpul compilării. Toate valorile variabilelor sunt determinate să se conformeze unui anumit tip.

Noțiunea de clasă este diferită față de noțiunea de tip. Ea conține două aspecte: *lucru şi depozit*. Aspectul lucru înseamnă crearea de obiecte noi care aparțin unei clase, iar aspectul depozit înseamnă ataşarea la o clasă a unor obiecte care sunt instanțe ale clasei. Utilizatorul poate manipula depozitul aplicând operații la toate elementele clasei. Clasele nu sunt folosite pentru corecția programelor, dar pot fi folosite pentru crearea şi manipularea de obiecte. Noțiunea de clasă apare mult mai frecvent în sistemele actuale de gestiune a bazelor de date orientate obiect.

O clasă este o construcție de limbaj folosită cel mai adesea la definirea tipurilor de date abstracte în limbajele de programare orientate pe obiecte. O clasă incorporează atât definiția structurii cât şi operațiile tipului de date abstracte. Elementele care aparțin colecției de obiecte descrise de o clasă sunt denumite instanțe ale clasei. O definiție minimală de clasă cuprinde următoarele:

- *Numele clasei*;
- *Operațiile externe* pentru manipularea instanțelor clasei. În mod tipic, aceste operații au un obiect țintă şi un număr de argumente. Operațiile de interfață sunt denumite metode ale clasei;
- *Reprezentarea internă*, care captează valorile diverselor stări ale instanțelor clasei, în variabile de instanță.

O definiție de clasă trebuie să includă şi codul care implementează operatorii de interfață ai clasei, ca şi descrierile reprezentării interne a obiectelor (stările obiectelor) din acea clasă. Valorile variabilelor din reprezentarea internă a instanțelor clasei aparțin obiectelor. Deşi valorile variabilelor din reprezentarea

internă diferă de la o instanță a clasei la alta, toate instanțele își partajează codurile care implementează operatorii de interfață. Aceştia au un scop similar cu apelurile de proceduri din limbajele de programare structurate.

1.5 Moştenirea

Moştenirea are două mari avantaje:

- Este o unealtă puternică de modelare, pentru că oferă o descriere concisă şi corectă a mediului;
- Ajută în partajarea specificațiilor şi implementărilor în aplicații.

În limbajele de programare orientate obiect, *moştenirea* se utilizează pentru partajarea operațiilor între clase care au aceleaşi caracteristici comportamentale. În modelele semantice, accentul este pus pe memorarea structurii, iar moştenirea se foloseşte pentru partajarea caracteristicilor fizice ale unei clase. În sistemele de baze de date orientate obiect, față de moştenirea specifică limbajelor de programare orientate obiect, trebuie combinate prin descriere şi memorare, atât caracteristicile structurale, cât şi cele comportamentale ale aplicației. Aceasta înseamnă că, toate operațiile definite pentru o clasă sunt valabile în orice subclasă a clasei date, iar fiecare instanță a unei subclase date este de asemenea instanță pentru oricare clasă la care aparține subclasa considerată.

În [Atkin 89] se arată că există patru tipuri de moştenire:

- *Moştenirea substituită,* care se bazează pe comportamentul moştenirii, nu pe valoarea sa;
- *Moştenirea inclusă,* care corespunde noțiunii de clasificare;
- *Moştenirea constrânsă,* care este o subclasă a moştenirii incluse;
- *Moştenirea specializată,* în cazul în care tipul t este subtip al tipului t'.

Există sisteme de baze de date orientate obiect în care este posibilă existența mai multor superclase pentru aceeaşi clasă. Această caracteristică se numeşte *moştenire multiplă*. Dacă anumite superclase definesc o proprietate identică, acest lucru poate determina ambiguitatea moştenirii, care trebuie obligatoriu soluționată funcție de sistemul considerat, prin introducerea unei definiții noi a proprietății în subclasa corespunzătoare.

Specializarea şi generalizarea claselor sunt alte două concepte strâns legate de conceptul de moştenire [Rob 95]. O clasă care moşteneşte de la o altă clasă, caracteristici structurale şi comportamentale devine *mai specială* prin adăugarea unor noi variabile instanță sau metode la nivelul acelei clase (figura 1.5).

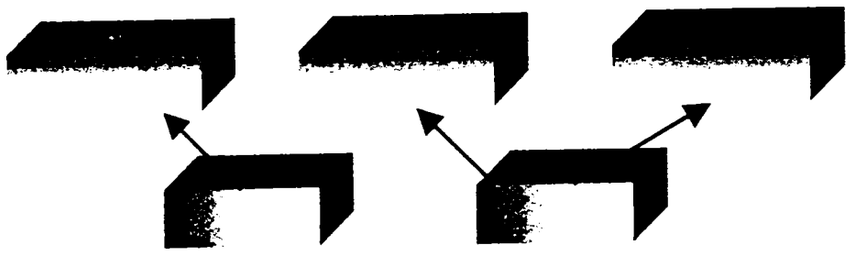


Fig 1.5 Specializarea claselor

Generalizarea este un mecanism care permite crearea de noi entități, pornind de la cele existente. Noile clase create sunt superclase ale celor existente (figura 1.6). Procesul de generalizare este unul de descoperire, în sensul că, numai după un anumit timp putem descoperi, dacă există o posibilă superclasă a unei clase date şi care sunt proprietățile şi metodele acesteia.

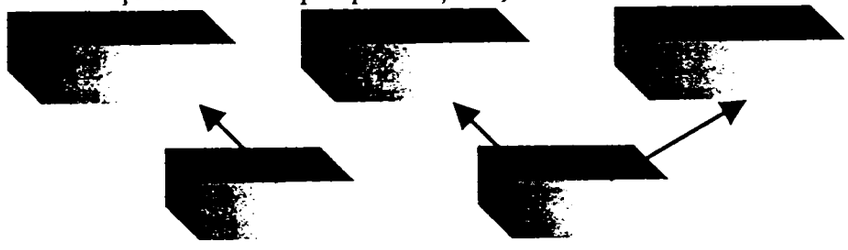


Fig 1.6 Agregarea claselor

1.6 Redefinire statică, polimorfism şi legare dinamică

Cel mai utilizat dintre cele trei concepte, este *supraîncărcarea* sau *redefinirea* şi el apare mai ales în limbajele de programare orientate obiect. Redefinirea este de două feluri:

- *Redefinire statică* (overloading), care are loc în faza de compilare;
- *Redefinire dinamică* (overriding), care are loc în faza de execuție.

Un sistem suportă *supraîncărcarea* (overloading) dacă este posibil ca, pentru clase diferite, să existe proprietăți cu nume identic. Din alt punct de vedere, acest concept permite apelarea operațiilor cu acelaşi nume, dar cu semnificații şi implementări diferite. Ea include operatorii aritmetici, operațiile de I/O, funcțiile de creare de obiecte şi operatorii de asignare de valori. Un alt concept, asociat îndeaproape cu supraîncărcarea, este cel de redefinire dinamică sau polimorfism (overrriding). Prin acest concept, sistemul ataşează în momentul execuției şi nu în momentul compilării, selectori de mesaje la metodele pe care le implementează. Polimorfismul (overriding) înseamnă că sistemul oferă o situație în care definirea unei operații într-o clasă să fie identică cu definirea altei operații dintr-o subclasă a clasei considerate (figura 1.7).

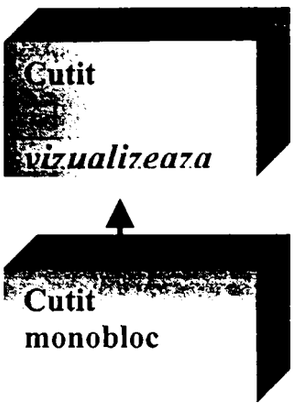


Fig 1.7 Polimorfismul

Într-un sistem de baze de date orientate obiect, care permite polimorfismul, o operație dintr-o subclasă poate avea același nume cu o operație a superclasei (sau a mai multora) de care aparține. Operația din subclasă se suprapune pe definiția operației din superclasă.

Consecințele polimorfismului sunt, în parte, adresate conceptului de legare întârziată (late building) sau legere dinamică. În sistemele de baze de date orientate obiect, comportamentul standard al operațiilor suprapuse, ne arată că, definiția aleasă pentru invocare, se sprijină pe clasa instanței asupra căreia se aplică operația. Într-un sistem care permite legarea întârziată, când o operație este invocată asupra unui obiect, sistemul caută în clasa obiectului respectiv definiția corespunzătoare din momentul apelului, iar dacă acea definiție nu există, va fi căutată în superclasele corespondente, până când este găsită. Acest comportament al sistemelor de baze de date orientate obiect este foarte utilizat în practică şi permite unei clase să se specializeze în comportamentul superclaselor.

La nivel abstract, *conceptul de legare* (binding) se referă la conexiunea logică dintre o entitate şi o proprietate a acesteia. În cazul programării orientate obiect, prin legare se înțelege corespondența dintre un mesaj transmis unui obiect şi rutina efectivă care se execută ca şi răspuns la acest mesaj. Legarea metodelor se împarte în două categorii:

- *Legarea statică* sau legarea timpurie (early-binding), în care rolul important îl are compilatorul şi editorul de legături;
- *Legarea dinamică* (late-binding), în care fixarea adresei de apel se realizează în momentul rulării. Compilatorul pregăteşte doar un tablou cu adrese posibile de apel. În momentul rulării, funcție de un pointer la o clasă de bază şi la un obiect de clasă derivată, se alege adresa de apel dorită.

Marele avantaj al legării dinamice, față de legarea statică, constă în faptul că, oferă un grad minim de flexibilitate. Astfel, ierarhii de clase sunt mai uşor de implementat şi totodată mai uşor de exploatat de la nivelul unui modul client. Dezavantajul legării dinamice față de legarea statică constă în diminuarea performanțelor de viteză.

Conceptele prezentate până în acest punct se referă în principal la programarea orientată obiect. Pentru un sistem de baze de date obiectuale [Atkin 89] consideră necesară următoarelor concepte: persistența, refacerea, concurența, interogări orientate obiect.

1.7 Completitudine computațională, extensibilitatea, persistență

Din punct de vedere al unui limbaj de programare, *completitudinea computațională* este evidentă: folosind limbajul DML al unui sistem de baze de date, se poate exprima orice funcție computațională. Din punct de vedere al bazelor de date, aceasta constitue o noutate. *Completitudinea computațională* poate fi introdusă printr-o legătură rezonabilă cu limabajele de programare.

Un sistem de baze de date are, în general, un sistem predefinit de tipuri. Aceste tipuri pot fi folosite de către programatori pentru proiectarea şi realizarea aplicațiilor dorite. *Extensibilitatea* setului de tipuri poate fi înțeleasă în două sensuri:

- În definirea noilor tipuri;
- Nu trebuie făcută distincție între tipurile sistem şi cele folosite de către utilizator.

În general, există o mare diferență între tipurile sistem şi cele definite de către utilizatori, dar din punct de vedere al aplicațiilor acest lucru trebuie să fie invizibil.

Peristența este evidentă, din punct de vedere al bazelor de date şi un lucru deosebit, din punct de vedere al limbajelor de programare. *Persistența* este abilitatea programatorului de a face ca datele aplicației să supraviețuiască cât mai mult în urma unui anumit proces, eventual să poată fi folosite şi într-un alt proces ulterior. *Persistența* poate fi *ortogonală*, fiecare obiect independent de tipul său este permis să devină persistent. Datele pot fi temporare şi persistente. Datele temporare există doar pe parcursul unui singur program sau a unei tranzacții. Datele persistente există în memorie pe parcursul mai multor tranzacții sau programe. Persistența ortogonală se bazează pe trei principii fundamentale:

- *Independența de persistență,* ceea ce inseamnă că persistența unui obiect este independentă de modul în care un program manipulează un obiect de date tot aşa cum un fragment de program este exprimat independent de persistența datelor cu care lucrează;
- *Ortogonalitatea tipurilor de date,* adică toate obiectele de date trebuie să posede acelaşi tip de persistență, nu unele să fie temporare şi altele să aibă viață lungă;
- *Persistența tranzitivă* este principiul prin care se identifică persistența furnizând obiecte de date persistente la nivelul limbajului

Persistența poate fi implementată de către programator sau poate fi o caracteristică a sistemului bazei de date. Un obiect persistent creat de către programator este creat având asociat un identificator unic şi permanent. În cazul

în care *persistența* este o caracteristică a sistemului, ea poate fi implementată în două moduri: prin moştenire şi prin referință [Atkin 89].

Persistența prin moştenire consideră persistența ca o proprietate moştenită de la o clasă rădăcină. Această clasă dispune de metode de creare şi distrugere de obiecte persistente.

Persistența prin referință asociază proprietatea de persistență direct obiectului, nu clasei căreia îi aparține.

1.8 Concurența

Concurența este un concept care se referă în general la activități care se desfăşoară în acelaşi timp şi care conduc la acelaşi rezultat ca şi când ar fi executate serial într-o anumită ordine. Conform [Khosh 93] există două categorii de algoritmi care controlează concurența: *algoritmi pesimişti* şi *algoritmi optimişti.*

Algoritmii pesimişti pleacă de la premiza că activitățile concurente intră în conflict una cu alta şi de aceea obiectele sunt blocate, adică ele devin inaccesibile pentru alte activități (tranzacții). Ei utilizează blocările pentru a sincroniza execuția activităților concurente. Algoritmii pesimişti de control al concurenței prezintă avantajul că elimină conflictele care pot să apară la nivelul sistemului, dar faptul că blocarea obiectelor este o mare consumatoare de timp, precum şi faptul că este necesară existența unui mecanism de deblocare în cazul în care două sau mai multe activități se blochează una pe alta, afectează performanțele generale ale sistemului.

Algoritmii optimişti se bazează pe faptul că două sau mai multe activități cer accesul concomitent la acelaşi obiect. În prealabil nu se face nici un control asupra desfăşurării activităților, doar la sfârşit se foloseşte o procedură de validare care elimină activitățile care nu s-au desfăşurat corect.

1.9 Recuperarea

Din [Khosh 93], [Atkin 89] rezultă că recuperarea datelor este un aspect important pentru bazele de date orientate obiect şi se bazează pe faptul că trebuie reținută imaginea obiectului, înainte şi după o acțiune. În general posibilitățile de recuperare se referă atât la aspecte soft cât şi la aspecte hard.

1.10 Facilități de interogare obiectuale

Interogarea este folosită pentru a obține mulțimi de obiecte in baza de date. Interogările obiectuale trebuie să fie simple, adică să fie optime. Limbajele care realizează interogări obiectuale, trebuie să găsească druul optim la setul de obiecte dorit. Interogarea obiectuală, va fi tratată separat în capitolele următoare ale tezei, cu referire directă la mediile pur obiectuale şi cele relațional

obiectuale. Se va dezvolta prezentarea acestui concept, în contextul prezentării claselor derivate, a modului de definire şi al modului de gestionare a acestora.

1.11 Agregarea

Agregarea stabileşte relaţia dintre un obiect şi componentele sale. O clasă agregat, este definită ca o clasă, a cărei structură, constă din unul sau mai multe obiecte simple [Popov 98]. Agregările pot fi catalogate în trei mari categorii:

- Agregate fixe, care au o structură fixă;
- Agregate variabile, care constau dintr-un număr fix de elemente, dar variabil;
- Agregate recursive, în cazul în care tipul de recursivitate apare în structura clasei

În capitolul următor, vor fi prezentate în amănunt principalele caracteristici ale bazelor de date orientate obiect.

2

Sisteme de Gestiune ale Bazelor de Date Orientate Obiect (OODBMS)

Sistemele de gestiune ale bazelor de date (SGBD) sunt entități complexe care au particularități specifice funcție de modelul de bază de date pe care îl deservesc. Peste modelul obiectual al bazelor de date este funcțional Sistemul de Gestiune al Bazei de Date Orientate Obiect. OODBMS trebuie să fie un sistem de gestiune legat de modelul obiectual al bazelor de date.

Un OODBMS este un sistem de bază de date care are încorporat modele de date suficient de puternice pentru a gestiona obiecte complexe.

Evaluarea unui OODBMS include următoarele criterii:

- funcționalitatea
- utilitatea
- platforma pe care este implementată
- performanțele

Studiul *funcționalității* se face în scopul de a asigura caracteristicile corespunzătoare ale bazei de date în procesul de dezvoltare a acesteia. Aceasta include funcționalitatea bazei de date, din punct de vedere al controlului concurenței și al regenerării, precum și din punct de vedere al caracteristicilor obiectuale: moștenirea și versionarea. Fiecare evaluare va avea ca scop identificarea unui set de cerințe funcționale necesare OODBMS-ului.

Utilitatea este legată cu procesul de întreținere al bazei de date precum și cu procesul de dezvoltarea al aplicației legată de baza de date.

Cel mai ușor criteriu de evaluat este *platforma* pe care se implementează baza de date. În acest caz trebuie luate în considerare posibilitățile hard, soft și de rețea ale mediului în care va fi implementată baza de date. Trebuie luate în considerare procesele care se vor desfășura pe server, procesele aplicației client, procesele specifice administratorului de bază de date precum și uneltele de dezvoltare. Cerințele de rețea trebuie de asemenea luate în considerare.

Performanța poate reprezenta cel mai important criteriu de evaluare. Trebuie luate în considerare următoarele criterii: numărul de useri interactivi,

rata de acces la informațiile bazei de date, mărimea bazei de date, configurația hard și de rețea a bazei de date.

Proiectarea bazei de date obiectuale este un proces integrat procesului de proiectare al aplicațiilor legate de baza de date. Clasele de obiecte folosite de către limbajele de programare sunt clase folosite de către baza de date obiectuală. Un domeniu uzual de folosire a bazelor de date orientate obiect este domeniul CAD, CAM, CASE (Computer Aided Software Engineering). O caracteristică generală a acestor aplicații constă în faptul că ele folosesc date complexe în mod eficient. Alte domenii în care pot fi aplicate bazele de date orientate obiect este cele al automatizării fabricilor sau al societăților comerciale. Bazele de date obiectuale permit o unificare a paradigmelor referitoare la modelele de date și integrează toate aspectele arborescente ale acestora într-un mod uniform aplicându-le tuturor utilizatorilor bazei de date.

Tehnologia bazelor de date orientate obiect este o legătură între programarea orientată obiect și tehnologiile bazei de date.

Probabil că una intre cele mai semnificative caracteristici ale bazelor de date orientate obiect este aceea de a combina programarea orientată obiect cu tehnologia bazelor de date pentru a avea un sistem integrat (figura 2.1). Sunt mai multe avantaje care decurg din includerea definirii operațiilor cu definirea

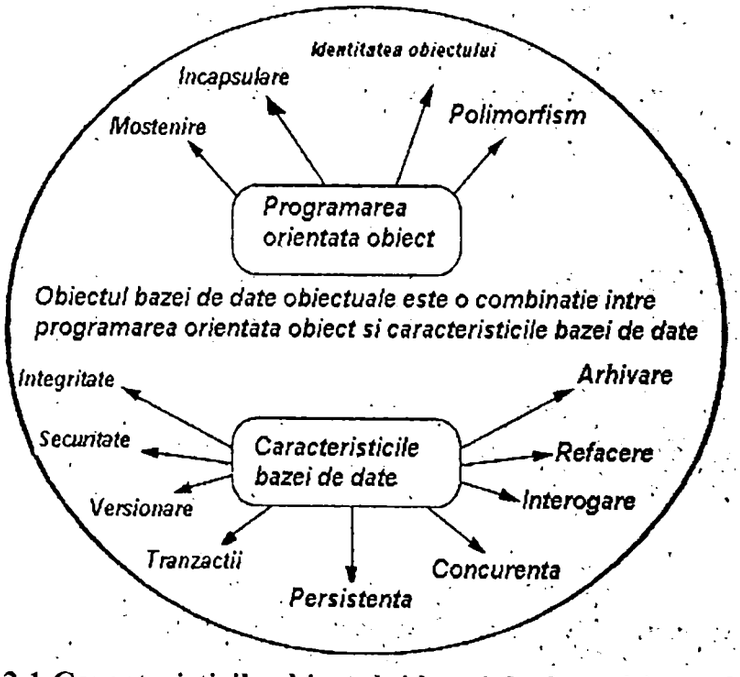


Fig 2.1 Caracteristicile obiectului bazei de date obiectuale

datelor. Primul avantaj constă în faptul că definirea operațiilor nu depinde în particular de o anumită aplicație, care se execută într-un anumit moment. Tipurile de date pot fi extinse pentru a suporta date complexe. Caracteristicile proprii limbajelor de programare orientate obiect au fost prezentate în capitolul precedent. În continuare vor fi prezentate principalele caracteristici ale bazelor de date orientate obiect, așa cum se vede în figura 2.1.

2.1 Principalele caracteristici ale OODBMS

2.1.1 Concurența

Acest concept permite lucrul concomitent al mai multor utilizatori la resursele unui sistem de baze de date orientate obiect. Pentru a preveni conflictele şi a păstra integritatea bazei de date, OODBMS-urile trebuie să dispună de *sisteme de control ale concurenței* şi de gestiune a tranzacțiilor. O tranzacție este o secvență de instrucțiuni dintr-un program, care citeşte sau scrie obiectele persistente ale bazei de date şi care are proprietăți de atomicitate, consistență, izolare şi durabilitate. Atomicitatea specifică faptul că, o tranzacție se execută complet sau nu se execută deloc. Consistența se referă la faptul că, toate restricțiile de integritate ale bazei de date sunt satisfăcute. Execuția tranzacției are ca efect trecerea bazei de date dintr-o stare de consistență, într-o altă stare de consistență. Izolarea se referă la execuția tranzacțiilor concurente, care manipulează în mod partajat aceleaşi obiecte din spațiul obiectual. Rezultatele intermediare din timpul tranzacțiilor concurente devin vizibile în exteriorul tranzacțiilor, numai după terminarea cu succes a acestora. Durabilitatea se referă la faptul că, actualizările tranzacțiilor terminate cu succes nu se pierd niciodată. Durabilitatea tranzacțiilor este strâns legată de recuperarea datelor în caz de incident. Acesate caracteristici vor fi prezentate pe larg în paragraful 2.1.5. Aşa cum s-a prezentat în capitolul 1, concurența este definită prin algoritmi pesimişti şi optimişti. În general algoritmii pesimişti utilizează blocările pentru a sincroniza execuția tranzacțiilor concurente. În [Stone 90] se arată că, blocarea obiectului, este cerută de către o tranzacție înainte de momentul în care doreşte să acceseze respectivul obiect. În general există două tipuri de blocări:

- blocarea pentru citire, se numeşte blocare *partajată*, care înseamnă că mai multe tranzacții pot executa operații de citire asupra aceluiaşi obiect din baza de date;
- blocarea pentru scriere sau citire sau blocarea *exclusivă,* care înseamnă accesul în scriere sau citire, la acel obiect, numai pentru prima tranzacție care a reuşit blocarea respectivului obiect.

Blocarea obiectelor conform [Stone 90] , se realizează printr-un protocol care se execută în două faze: prima fază prin care tranzacția sau tranzacțiile câştigă dreptul de a bloca obiectul respectiv şi faza a doua, prin care obiecte sunt deblocate, pentru a putea fi accesate de către alte tranzacții. În general sistemele de baze de date obiectuale, relațional obiectuale, relaționale, folosesc sistemul de blocare partajat sau exclusiv.

Sistemele de baze de date relațional obiectuale ca de exemple Visual Dbase, Visual Fox, Access, permit blocarea la nivel de tuplă a relației sau la

nivel de tabelă a bazei de date. În Visual Fox blocarea la nivel de tabelă se realizează cu comanda funcția FLOCK() iar la nivel de tuplă sau înregistrare prin RLOCK() dacă este vorba despre o singură înregistrare sau LOCK() dacă sunt mai multe înregistrări blocate. Deblocarea tabelei, sau a înregistrărilor se realizează cu funcția UNLOCK(). Alte comenzi adiacente sunt SET MULTIPLE LOCKS(), SET REPROCESS TO, etc.

Unele sisteme de baze de date pur obiectuale ca de exemplu Versant, Ontos, Jasmine, suportă pe lângă tipul de blocări prezentat, un tip mai special numit intenție de blocare. Acest tip de blocare se manifestă înainte ca să se realizeze o blocare de tip partajat sau în extensie.

Aşa cum s-a arătat blocarea poate să se manifeste la nivel de obiect dar poate să se manifeste şi la nivel de clasă (figura 2.2).

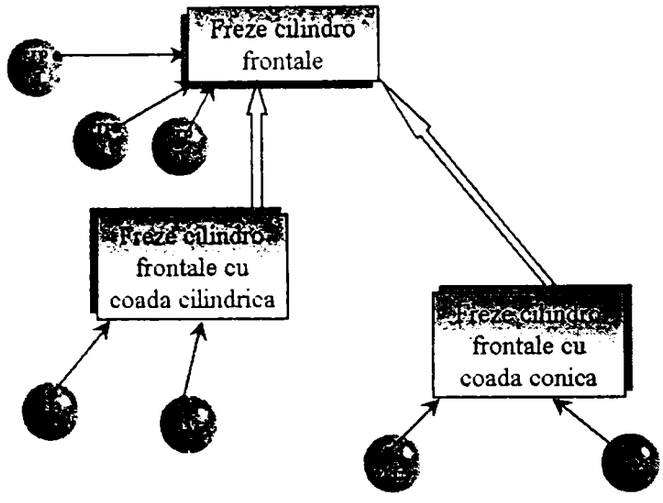


Fig 2.2 Blocarea la nivel de obiect şi de clasă

Fiecare nod din structura ierarhică poate fi blocat. Dacă se cere o blocare de tip exclusiv şi aceasta este garantată, atunci blocarea se va transmite subclaselor în structura ierarhică. De exemplu dacă se doreşte o blocare a clasei "Freze cilindro frontale" atunci blocarea se va transmite şi subclaselor "Freze cilindro frontale cu coadă cilindrică" şi"Freze cilindro frontale cu coadă conică". În cazul unei blocări de tip partajat lucrurile se desfăşoară asemănător. Orice nod al clasei poate fi blocat în modul intenție. Înainte de intenția de blocare a clasei toți strămoşii clasei respective trebuie blocați în intenție. Cu alte cuvinte se manifestă intenția de blocare asupra tuturor instanțelor claselor aflate în ierarhie. În cazul obiectelor complexe, obiectele independente, referite de către obiectul complex sunt blocate separat. Uneori o tranzacție poate dori să citească toate instanțele unei clase şi să modifice doar o parte din acestea. În acest caz se poate bloca cu intenție de citire-scriere o clasă, după care instanțele care se doresc modificate, se vor bloca în scriere.

O altă problemă care poate apare în cazul tranzacțiilor concurente este arătată în [Stone 90] este deadlock-ul sau mecanismul prin care două sau mai multe tranzacții se blochează una pe cealaltă în procesul de blocare a unui obiect. Din această cauză este necesar un mecanism suplimentar care să dedecteze şi să soluționeze deadlock-ul. În Jasmine este folosită o clasă specială denumită LISTENER, care are rolul de a minimiza posibilitatea de apariție a deadlock-ului. Aceasta se face prin evitarea actualizării instanțelor clasei LISTENER, în cazul mai multor tranzacții. Pentru evitarea deadlockului se folosesc subclase ale clasei Listener, care centralizează actualizările şi le actualizează folosind tranzacții separate.

2.1.2 Refacerea

Acest concept se referă la faptul că, un anumit sistem de baze de date orientat obiect trebuie să ofere posibilități de refacere sau de recuperare a informației, în cazul în care apar erori din punct de vedere hard şi soft. Există trei tipuri de erori care pot apărea:

- *Erori la nivelul tranzacțiilor*, apar în cazul în care o tranzacție nu este terminată corect, datorită unor erori care apar datorită concurenței sau datorită abandonului tranzacției de către utilizator;
- *Erori datorită sistemului,* care pot fi de natură hard sau soft, şi se referă atât la erori care apar datorită softului care gestionează baza de date orientată obiect, cât şi datorită sistemului de operare. Totodată acest tip de erori, pot apare datorită difuncționalităților din punct de vedere fizic a sistemului ;
- *Erori datorită memoriei externe* (hardisk) sunt cele mai greu de rezolvat, şi sunt cauzate de erorile care le generează porțiuni din suportul de stocare al datelor sau suportul în ansamblul său.

Pentru a putea rezolva aceste erori ce pot apărea în baza de date, există un modul special, în baza de date care rezolvă aceste probleme, denumit managerul de erori. Există diverse structuri de date folosite pentru realizarea acestuia. Una dintre ele constă în folosirea *jurnalului de arhivare* sau a *log-ului*,ce păstrează în timp imaginea obiectului înainte şi după actualizare. În cazul în care tranzacția se termină cu succes actualizările trebuie memorate pe disc. În acest caz noua stare a obiectului este scrisă pe disc şi în log. Dacă tranzacția nu s-a încheiat cu succes atunci vechea stare, adică cea care se găseşte în jurnalul de arhivare este scrisă pe disc.

Toate sistemele de baze de date orientate obiect asigură suport pentru refacerea datelor în caz de incident. De exemplu în Gemstone se permite administratorului bazei de date să creeze copii ale bazei de date pe disc. Sistemul Ontos dispune de un utilitar specializat Dbrecover care asigură

refacerea bazei de date şi reconstituirea consistenței acesteia. În Versant sistemul de refacere al datelor poate fi activat sau dezactivat. Dacă el este activat performanțele sistemului sunt micşorate în sensul că viteza de execuție devine mai mică. În Visual Fox există o serie de comenzi care controlează tranzacțiile şi refac informația în caz de incident. Acestea sunt SQLCOMMIT() returnează 1 dacă tranzacția s-a terminat cu succes, dacă nu se poate folosi AERROR() pentru a vedea cauza care a generat eroarea tranzacției. ROLLBACK anulează toate modificările făcute pe parcursul unei tranzacții. O comandă asemănătoare este SQLROLLBACK(). În Jasmine metoda ROLLBACK() a clasei TRANSACTION are rolul de a inhiba scrierea rezultatelor tranzacției, în baza de date.

2.1.3 Interogare

O *interogare* este o specificare declarativă a unei mulțimi de obiecte din baza de date. De obicei, condițiile sunt specificate printr-o combinație booleană de predicate, de forma <nume_atribut, operator de comparație, valoare>. Ea este formulată de obicei pentru o porțiune a schemei bazei de date.

O interogare trebuie să respecte o serie de criterii:

- să fie de nivel înalt;
- să fie capabilă să lucreze rapid;
- să pună accentul pe ceea ce face, nu cum face;
- să fie eficientă;
- să poată forma aplicații independente, care să poată fi rulate cu orice fel de tip de bază de date.

Dacă se ignoră ierarhia de clase şi se consideră că o interogare se referă la o singură clasă, aceasta va consta din regăsirea unei mulțimi de instanțe ale clasei care satisfac condițiile cerute de către utilizator. Deoarece o instanță a unei clase este un obiect format din instanțe ale domeniilor atributelor sale, o interogare referitoare la o clasă este, de fapt, o interogare referitoare la clasă şi la unele domenii ale atributelor clasei. Într-o bază de date, atributele pot fi simple sau complexe. Un atribut simplu este un atribut al cărui domeniu este o clasă primitivă sau o clasă de bază. Un atribut complex are ca domeniu o clasă care, la rândul său, are mai multe atribute simple sau complexe. Un model de interogări pentru bazele de date orientate obiect, a fost realizat în [Baner 88] . În sistemele orientate obiect, interogările pot fi evaluate într-o manieră similară celor relaționale. Echivalentul operației de selecție a bazelor de date orientate obiect este simpla regăsire a instanțelor clasei, iar regăsirea acestor instanțe, cere regăsirea instanțelor altor clase, care sunt referite recursiv ca valori pentru atribute.

ODBMS-urile suporă diverse forme de interogări, acest lucru se datorează faptului că un ODBMS suportă un anumit tip de model obiect. În unele cazuri

ODBMS-urile suportă extensiile de clase ca şi colecții care sunt interogate. În alte clazuri extensiile de clase nu sunt suportate şi pot fi interogate, doar colecții de obiecte. În ambele cazuri cele două tipuri de structuri pot fi interogate. În unele sisteme pot fi interogate, atât obiectele persistente cât şi cele temporare pe când în altele nu pot fi interogate decât cele persistente. ODBMS-urile pot diferi şi prin rezultatele pe care le produc interogările. Într-un DBMS relațional rezultatul interogării este definit într-o altă relație, ea fiind definită ca parte a modelului relațional. Proprietatea de închidere (closure) asigură că rezultatele interogării au fost bine definite. Din acest punct de vedere unele interogări implementate în ODBMS lasă de dorit. În general rezultatele unei interogări trebuie să fie:

- o mulțime de obiecte existente;
- o mulțime de obiecte noi ale unor clase existene;
- o mulțime de obiecte noi fără identificatori;
- o mulțime de obiecte noi ale unor clase noi.

La momentul actual limbajele de interogare obiectuale au o serie de limitări care se ilustrează în modul în care pot fi construite noi obiecte precum şi a tipurilor sistem pe care le suportă. ODBMS-urile diferă şi prin modul în care limbajul de interogare este integrat cu unele limbaje de programare. Unele suportă variante SQL (ONTOS, VERSANT), altele suportă limbaje de interogare integrate în C++ sau Smalltalk (în ObjectStore, GemStone, Jasmine).

Un exemplu de interogare folosind ODQL, programul de interogare din mediul Jasmine integrat în C++, este următorul:

```
List <freze> pp;
Pp = select from freze where freze.codf=10001
```

O altă variantă permite crearea unui limbaj de interogare independent care poate fi folosit direct ca limbaj de programare (OpenODB şi SQL3).

Un exemplu de program scris în SQL3 este următorul:

```
create type freza (nume char(20), cod_freza integer, tip char(20))
create type reper (nume_rep char(20), cod_rep integer, freza_ap freza)
create type freza_cil_front_cu_placute under freza
        (D integer, dH7 integer, L integer, b integer, t integer, L integer, l integer, z integer,
reper_plac reper)
create type placuta under reper (cod_plac integer, codf freza, nume)
create table freze of freza (primary key (cod_freza))
create table frezecilfrontplac of freza_cil_front_cu_placute
create table placute of placuta
.....
Select nume, cod_plac
        From placuta
        Where codf.nume="freze cilindro frontale cu placute" and reper.cod_rep=1000
```

Aceste forme diferite de interogări afectează nu numai modul în care facilitățile de interogare apar la utilizator dar afectează şi mecanismele interne care suportă procesele de interogare.

Optimizarea interogărilor obiectuale, are ca scop găsirea celui mai bun drum către obiectele bazei de date care participă la interogare. De regulă există

mai multe căi spre acestea, rolul optimizatorului fiind acela de a evalua pe fiecare şi de a găsi metoda optimă.

Într-o bază de date obiectuală, un atribut poate fi simplu sau complex. Este simplu în cazul în care domeniul său este o clasă de bază şi este complex în cazul în care domeniul său este o clasă, care are la rândul său alte atribute simple sau complexe. Conform cu [Khosh 93] , o clasă împreună cu clasele cărora aparţin atributele complexe sunt reprezentate sub forma unui graf, numit graf de interogare figura 2.3. Evaluarea interogării presupune întocmirea unui plan de evaluare. Optimizarea se bazează pe luarea în considerare a costurilor fiecărui plan de evaluare prin traversarea grafului înainte sau înapoi. Pentru regăsirea obiectelor bazei de date care satisfac căutarea cât mai eficientă, se folosesc diverse structuri de memorare, ca de exemplu indecşii.

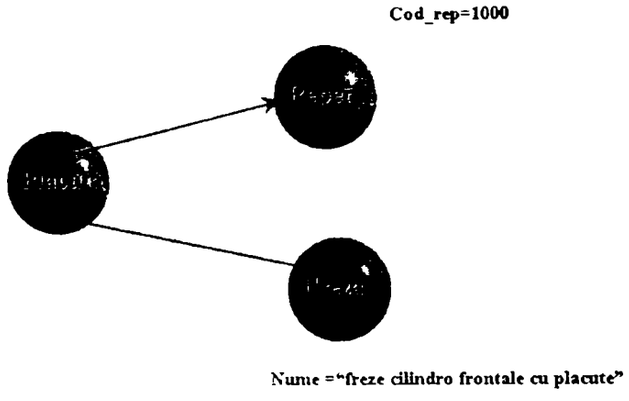


Fig 2.3 Graf de interogare

Optimizarea interogărilor obiectuale, depinde de abilitatea cu care sunt definiţi indecşii, pe un atribut la nivelul unei singure clase sau la nivelul unui atribut, la nivelul tuturor claselor care aparţin grafului de interogare. De asemenea ea depinde de structuri auxiliare implementate la nivel înalt, care cer date sau obiecte. Cele mai multe cercetări în domeniul optimizării interogărilor au fost făcute în domeniul optimizării interogărilor relaţionale. Aceste cercetări se extrapolează în prezent spre interogările obiectuale. În modelul relaţional sunt fixate un set de operaţii peste o structură realativ simplă care este relaţia normalizată. Aceste structuri au un set relativ simplu de metode de acces definite la implementarea DBMS-ului. Spre deosebire de acestea ODBMS-urile sunt extensibile, adică pot fi adăugate sistemului în orice moment tipuri de date arbitrare şi operaţii asociate acestora. De asemenea pot fi implementate metode care suportă aceste tipuri de date. Interogările obiectuale pot conţine combinaţii arbitrare între fiecare operaţie definită de către utilizator. Fiecare tip nou introduce o operaţie nouă a cărei proprietăţi nu sunt cunoscute de către optimizatorul interogării. Fără cunoaşterea noilor operatori sau a proprietăţilor

lor nu se poate vorbi despre optimizarea interogării. Mai mult decât atât, în ODBMS memorarea şi implementarea metodelor de acces a obiectelor sunt încapsulate. Datorită acestor aspecte ODBMS-ul foloseşte tehnici complexe de indexare, tehnicile de clustering jucând de asemenea un rol deosebit de important. Concluzionând optimizarea ODBMS-ului trebuie să fie complexă şi extensibilă. Informațiile privind optimizarea interogărilor obiectuale provin din mai multe surse. O parte se referă la operațiile ce vor fi îndeplinite. Acestea include metode definite pentru clase utilizator sau tot atât de bine operații built in, definite pentru clase de obiecte ale ODBMS-ului. Informațiile despre metodele obiectelor pot fi obținute prin analiza expresiilor limbajului de programare care definesc metode obiect pentru determinarea oportunităților de optimizare. De aceea în general ODBMS-ul restricționează optimizarea la limbajul de interogare. Aceasta poate fi o variantă SQL sau sintaxe de interogare specializate, adăugate unui limbaj de programare (extensii C++ în ObjectStore sau extensii Smalltalk în GemStone).

2.1.4 Versionare

Versionarea este un concept care permite urmărirea schimbărilor întâmplate în starea unui obiect. Versionarea este datorită acestui fapt, un concept foarte puternic, în special în CAD, CAM. La nivelul acestor domenii, obiectele suferă mai multe transfomări, astfel putem vorbi de mai multe versiuni ale aceluiaşi obiect. Versionarea are drept scop crearea uneltelor necesare pentru manipularea diverselor versiuni ale aceluiaşi obiect. De obicei, versiunile obiectelor, în general complexe, sunt păstrate în containere numite *repository*. Pot fi create versiuni seriale sau paralele ale aceluiaşi obiect (fig 2.4).

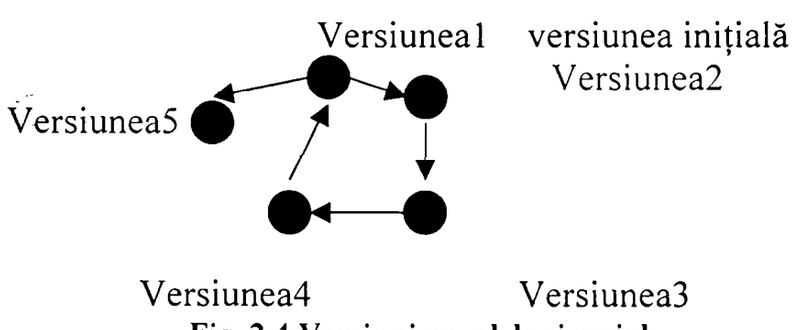


Fig. 2.4 Versiuni paralele şi seriale

Versiunea 1 se consideră versiunea inițială, versiunea 5 este versiunea finală. Versiunile 2 şi 5 sunt *versiuni paralele sau alternative* ale versiunii 1. Versiunile 1,2,3,4 sunt *versiuni seriale sau liniare*. Cu alte cuvinte dacă versiunile sunt create secvențial, aceasta se mai numeşte *versionarea liniară*, dacă versiunile sunt create în paralel, se obține o *versionare alternativă*. De asemenea versionarea poate permite noilor versiuni să fie create prin returnarea unei

versiuni mai timpurii şi crearea de versiuni derivate. Operaţiile permise în versionare sunt de creare şi ştergere versiuni, creare şi ştergere versiuni derivate, îmbinare de versiuni, definirea unor versiuni specifice sau implicite. Procesul de realizare al versiunilor de date, poate fi folosit pentru mărirea concurenţei, deoarece diverşi proiectanţi pot lucra concomitent la mai multe versiuni ale aceluiaşi obiect. În unele medii orientate obiect există trei tipuri de versiuni:

- *versiunea tranzitorie* care este instabilă şi nu pot fi reactualizate sau şterse;
- *versiunile de lucru* care sunt considerate stabile şi nu pot fi reactualizate, dar ele pot fi şterse de cei care le crează;
- *versiunile puse în circulaţie* sunt considerate stabile şi nu pot fi reactualizate sau şterse.

În mediul Versant, versionarea este atât liniară cât şi alternativă. În figura 2.5 este prezentat un exemplu de versionare în Versant. Versiunea V1 este versiunea curentă. Ea are prioritate în faţa versiunilor alternative, V2. Este posibil ca versiunile să fuzioneze, sau dacă apare un incident, se păstrează versiunea V1.

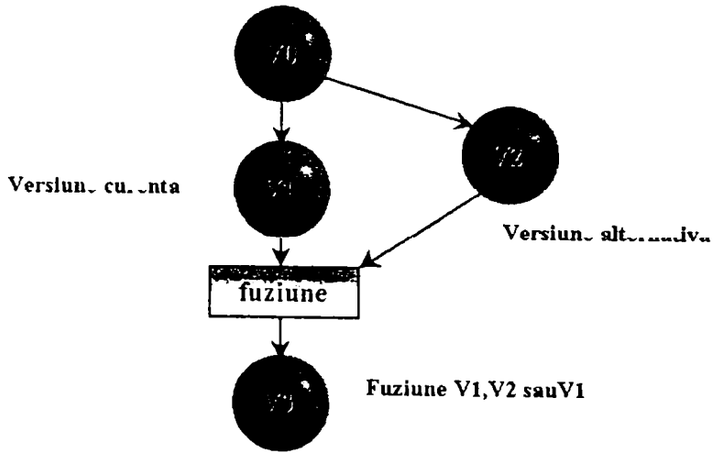


Fig 2.5 Versionarea în Versant

În Visual Fox, versionarea se realizează cu version engine, care forţează execuţia celei mai recente versiuni de program sau se pot crea mai multe versiuni ale bazei de date. Proprietăţile versiunii bazei de date se pot seta cu comanda DBSETPROP(), iar consultarea acestora se poate face cu ajutorul funcţiei DBGETPROP().

2.1.5 Tranzacţiile în bazele de date orientate obiect

În majoritatea modelelor bazelor de date este prevăzut conceptul denumit *tranzacţie*. În modelul obiectual, aceasta este o secvenţă de acţiuni care pot citi sau scrie obiecte într-o bază de date şi satisfac testul ACID adică proprietăţile de *atomicitate*, *consistenţă*, *izolare* şi *durabilitate*.

Asupra obiectelor pot acționa, în același moment, mai multe tranzacții. Acestea se numesc concurente. Ele nu trebuie să interfereze unele cu altele.

O tranzacție care se termină cu succes nu trebuie niciodată pierdută. De aceea, chiar în cazul în care apar erori de sistem, este necesară salvarea datelor, în cazul în care tranzacția s-a terminat cu succes.

Proprietățile ACID permit folosirea în siguranță a datelor partajate. Fără aceste proprietăți orice eveniment care apare în sistem poate genera efecte neprevăzute. În special în aplicațiile de e-commerce aceste proprietăți sunt foarte importante în sensul în care se poate cumpăra de exemplu obiectul dorit.

2.1.5.1 Atomicitatea

Atomicitatea se referă la faptul că o tranzactie poate fi executată sau în întregime, sau deloc. Aceasta înseamnă că, o tranzacție începută, fie se termină corect, fie nu se execută deloc în caz de incident. Tranzacțiile care prezintă erori trebuie eliminate complet din bazele de date obiectuale.

Principalele implicații ale atomicității sunt:

- Tranzacția este o operație unitară adică toate etapele unei tranzacții se desfășoară normal sau nici una nu se desfășoară.
- Atomicitatea este menținută în prezența timpilor morți (incidentelor).
- Atomicitatea este mentinută în prezența unor erori soft ale bazei de date.
- Atomicitatea este mentinută în prezența unor erori soft ale aplicației.
- Atomicitatea este mentinută în prezența unor erori ale procesorului
- Atomicitatea este mentinută în prezența unor erori ale discului.
- Atomicitatea poate fi implementată la nivel de sistem sau la nivel de sesiune.

2.1.5.2 Consistența

În general, o bază de date trebuie să respecte diverse reguli de integritate. O tranzacție nu poate să violeze aceste reguli de integritate ale bazei de date. Aceasta implică faptul că, o tranzacție trebuie să transfere baza de date dintr-o stare care respectă regulile de integritate (consistentă), într-o altă stare care respectă regulile de integritate (consistentă), adică, cu alte cuvinte, ea trebuie să fie *consistentă.*

În momentul în care baza de date este accesată concomitent de către mai mulți utilizatori sau de către mai multe aplicații, problema menținerii integrității datelor devine critică. *Consistența* face posibil acest lucru. Consistența poate fi privită ca un factor care menține informația dorită de către fiecare utilizator al

unei baze de date fără să se perturbeze unul pe celălalt. Gradele consistenței sunt:

Gradul 0 tranzacția nu rescrie data actualizată de către alt utilizator sau proces a altei tranzacții.
Gradul 1 este gradul 0 plus faptul că o tranzacție nu execută alte scrieri până când ea nu s-a terminat
Gradul 2 este gradul 1 plus faptul că o tranzacție nu foloseşte datele unei alte tranzacții.
Gradul 3 gradul 2 plus faptul că o tranzacție nu citeşte date noi înainte ca tranzacția să se încheie.

2.1.5.3 Izolarea

Tranzacțiile concurente sunt acele tranzacții care apar în acelaşi timp.

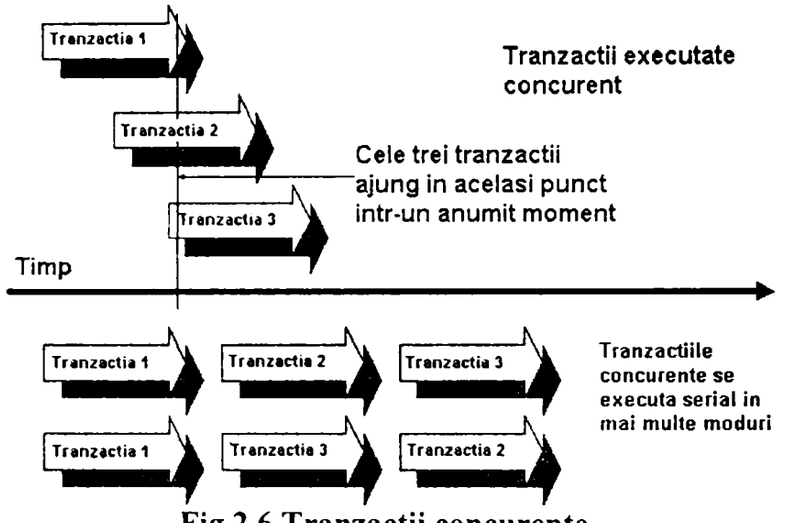


Fig 2.6 Tranzacții concurente

Aceste tranzacții sunt ilustrate la începutul figuri 2.6. Conceptul de siguranță care previne conflictele între tranzacții concurente se numeşte *izolare,* sau cu alte cuvinte, rezultatelor tranzacțiilor concurente ce acționează asupra aceluiaşi obiect sau grupuri de obiecte trebuie să fie protejate.

Un concept important pentru a înțelege *izolarea* prin tranzacții este *serializarea.* Tranzacțiile sunt serializate când efectul în baza de date este acelaşi, când tranzacțiile se execută în mod serial una după cealaltă, sau se execută în mod intercalat. Aşa cum se vede în figura 2.6, Tranzacția1 până la Tranzacția3 sunt executate concurențial tot timpul. Efectul în DBMS, este acela că, tranzacțiile pot fi executate în mod serial, bazându-ne pe consistență şi izolare. În partea de jos a figurii sunt prezentate diverse moduri în care tranzacțiile pot fi executate. Este important de relevat faprul că execuția serială nu implică prima tranzacție care trebuie să se execute.

2.1.5.4 Durabilitatea

Menținerea actualizărilor în tranzacțiile încheiate este critică. Aceste actualizări nu pot fi pierdute Durabilitatea realizează acest lucru. Durabilitatea este caracteristica sistemului de a recupera actualizările din tranzacțiile închise datorită unor erori ce pot apărea în sistem. Proprietăți ale durabilității sunt

- Recuperarea celei mai recente tranzacții reuşite până în momentul în care apare o eroare soft în baza de date.
- Recuperarea celei mai recente tranzacții reuşite până în momentul în care apare o eroare soft într-o aplicație.
- Recuperarea celei mai recente tranzacții reuşite până în momentul în care apare o eroare de procesor.

În general, în bazele de date orientate obiect, o tranzacție este cod program scris între cuvinte rezervate de tipul *begin transaction* şi *end transaction.*

În Visual Fox începutul tranzacției este marcat de către BEGIN TRANSACTION iar sfârşitul prin END TRANSACTION. În cazul în care nu intervin erori modificările efectuate devin permanente în baza de date. Sfârşitul tranzacției poate fi precizat folosind funcția SQLCOMMIT(). În cazul în care tranzacția nu se termină corect, este necesară revenirea la starea inițială a bazei de date, folosind comanda ROLLBACK sau SQLROLLBACK(). Nivelul unei tranzacții, este returnat de către funcția TXNLEVEL().

Limbajul ODQL al mediului Jasmine tratează tranzacțiile prin comenzi specifice. Pot fi folosite metode de lansare a comenzii (transaction.start()), de sfârşit a acesteia (transaction.end()) sau de tratare în caz de incident prin recuperarea bazei de date (transaction.rollback()). În următorul exemplu este prezentată o tranzacție scrisă în mediul Jasmine:

```
Transaction.start()    început tranzacție
Bag<classA> xs;
ClassA x;
xs = classA  from classA   } tranzacția
scan(xs,x) {
        x.print()
};
transaction.end()      sfârşit tranzacție
```

Un avantaj esențial al mediilor orientate obiect, este acela că se pretează mai bine pentru tratarea aplicațiilor CAD, CAM, care necesită tranzacții îndelungate de ordinul orelor sau chiar al zilelor. În acest caz apariția unor erori şi reluarea de la început a tranzacției eşuate, ar necesita un timp mult prea îndelungat pentru desfăşurarea unei aplicații. O primă soluție ar fi serializarea sau "spargerea" unei tranzacții mari, în mai multe tranzacții mai mici cu posibilități de reluare a acestora. Astfel în caz de incident, se vor relua doar tranzacțiile mici, fără a fi necesară reluarea tranzacției de la început (figura 2.7).

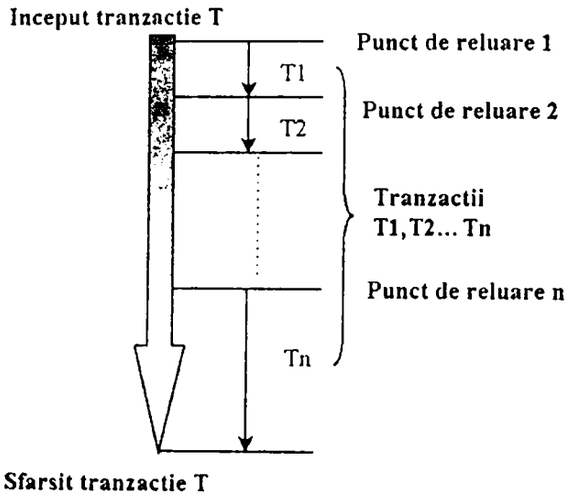


Fig 2.7 Serializarea tranzacțiilor

În Visual Fox tranzacțiile sunt *imbricate* pe 5 nivele. O altă abordare a tranzacțiilor mari este conceptul de imbricare a lor, adică descompunerea tranzacției într-o structură asemănătoare cu structura ierarhică. Există o tranzacție la nivel superior care este rădăcina, si mai multe tranzacții fiu, organizate pe unul sau mai multe nivele. Într-o tranzacție imbricată, toate tranzacțiile de pe nivel inferior trebuie să se încheie cu succes, pentru ca tranzacția de pe nivelul superior să se încheie cu succes (figura 2.8)

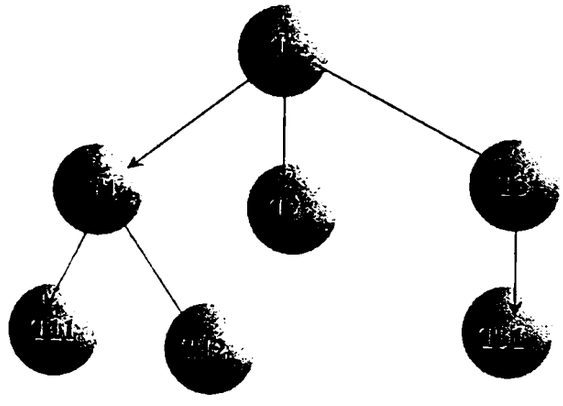


Fig 2.8 Tranzacții imbricate

Dacă la nivelul inferior apare o eroare într-o tranzacție, atunci tranzacția părinte încearcă reluarea tranzacției fiu până când aceasta se termină cu succes sau poate abandona execuția întregii tranzacții sau poate continua execuția fără tranzacția fiu producând un rezultat parțial.

Sistemele Versant, ObjectStore suportă un alt tip de tranzacții şi anume *tranzacțiile cooperante*. Acestea se bazează pe mecanismul *chack-out/check-in.*

Prin mecanismul check out mai mulți utilizatori care partajează aceeași bază de date, pot descărca obiectele de care au nevoie într-o bază de date proprie a lor, unde să manipuleze respectiva bază de date la nivelul unei tranzacții. După terminarea tranzacției, se vor face vizibile rezultatele acesteia, la nivelul bazei de date partajate, mecanismul check-in. Prin acest mecanism de folosire a unei baze de date proprii utilizatorului, se micşorează traficul în rețea. Obiectele bazei de date care vor fi descărcate de către un utilizator sunt blocate pentru ceilalți utilizatori. Acest lucru implică faptul că decizia dacă o tranzacție s-a terminat sau nu corect revine utilizatorului, care încalcă proprietatea de *izolare* a tranzacțiilor. Datorită acestui fapt acest tip de tranzacții nu este foarte utilizat.

2.1.6 Securitatea

Securitatea datelor este un concept deosebit de important care se caracterizează prin:

- Protejarea datelor de acțiunea unor factori externi (viruşi, utilizatori nedoriți);
- Asigurarea unor backup-uri în timp real datelor existente în bazele de date obiectuale;

În general bazele de date obiectuale au ca suport hard mai multe calculatoare legate în rețea. Implicit există mai mulți utilizatori care au acces la resursele bazei de date. Fiecare utilizator are însă acces la un număr limitat de resurse funcție de drepturile pe care i le asigură administratorul de rețea. Din acest punct de vedere utilizatorul trebuie să aibă *drepturi de acces* în baza de date şi să fie *identificat.* Din punct de vedere al *drepturilor de acces* utilizatorul are drepturi de citire sau scriere sau de citire-scriere, în anumite zone ale bazei de date, funcție de drepturile oferite de către administratorul de rețea. Aceste drepturi pot fi acordate sau revocate fiind specifice unor OODB-uri ca Versant, Visual Dbase Visual Fox, ONTOS. Din punct de vedere al identificării utilizatorului acesta primeşte o parolă de acces în baza de date de la administratorul de rețea. Această tehnică este disponibilă la nivelul tuturor OODB-urilor. *Auditarea* este un mecanism important şi el se referă la urma pe care o lasă un anumit utilizator în cazul în care accesează date de importanță majoră. Păstrarea "urmelor" se face într-un fişier denumit "audit trail" întreținut de către sistem. *Criptarea datelor* este un mecanism care se aplică în cazul în care obiectele persistente sunt salvate pe suporți de memorare sau sunt transmişi pe linii de comunicație, pentru a preveni accesul unor persoane nedorite.

Legarea bazei de date la rețeaua Internet presupune asigurarea unei securități suplimentare a datelor din punct de vedere al accesului extern şi a eventualilor viruşi care pot apare în rețea. Cel de al doilea aspect privind

securitatea datelor se materializează prin lucrul în oglindă cu mai multe servere sau trecerea pe un suport suplimentar al datelor la intervale regulate de timp.

2.1.7 Persistența

Datele existente într-o bază de date orientată obiect pot fi de două tipuri şi anume:

- *Temporare*, adică ele există atâta timp cât există o procedură sau o tranzacție;
- *Persistente* adică ele există pe suportul de memorare, şi sunt supuse actualizărilor.

Persistența este proprietatea datelor care determină, cât timp ele există, atâta timp cât sistemul de calcul este funcțional. Persistența poate fi:

- *Orogonală*, orice dată de orice tip poate persista oricând;
- *Transparentă*, programatorul poate trata persistent sau temporar obiectele;
- *Independenta* nu pot exista citiri şi scrieri explicite în baza de date.

Există mai multe aspecte ale persistenței în cadrul unui ODBMS.

Unul dintre aspectele persistenței ODBMS, este acela privind modul în care obiectele sistemului devin persistente. Un obiect persistent are asignat un identificator unic, care permite găsirea acestuia. De obicei se pleacă de la existența obiectului în spațiul temporal, adică în memorie şi se face legătura printr-un pointer cu spațiul de pe disc unde obiectul este persistent. Crearea obiectului persistent se poate face cu o operație de genul:

OID=Persistent(pointer)

Pointerii la obiecte în limbajele de programare sunt implementați prin adrese de memorie virtuală. Pentru obiectele persistente pot fi folosiți pointeri care nu sunt legați de o poziție fizică. Nu există nici o garanție că această poziție va rămâne constantă pe parcursul proceselor. De aceea majoritatea ODBMS-urilor generează identificatori de obiect care se găsesc în tabele şi care permit găsirea cu exactitate a obiectului dorit. În general se acționează asupra unui număr mai mare de obiecte. Gruparea fizică a obiectelor care vor fi accesate înseamnă *clustering*, care este un concept deosebit de important al sistemelor orientate obiect. Multe ODBMS-uri au prevăzute mecanisme de definire a clusterelor. Aceste procese sunt statice astfel încât un obiect este asignat la un cluster când este creat şi orice schimbare în clustring cere o reorganizare.

În *schema bazei de date*, ODBMS prevede existența spațiului separat (baza de date) pentru obiectele persistente. Clasele sunt declarate persistente şi toate instanțele claselor sunt persistente. Aplicațiile operează cu mulțimi de clase temporare, în memorie. Obiectele sunt definite ca persistente, prin crearea lor explicită în memorie şi sunt făcute nepersistente prin ştergerea lor din baza de date. Translația dintre obiectele persistente ale OODBMS-ului şi obiectele

temporare ale limbajului de programare trebuie să fie făcută de către programator schimbând starea acestora. Aceasta este similar cu interfața dintre limbajul de programare şi baza de date relațională. Dacă ODBMS-ul implementează metode în baza de date accesul la obiecte poate fi obținut prin aplicații obiect iar mesajele transmise de la aplicații la obiectele bazei de date să determine execuția metodelor obiectelor.

Un alt mod de a face obiectele persistente, este *instanțierea explicită persistentă sau prin referință*. În acest mod obiectele sunt făcute persistente prin proiectarea explicită a lor, ca persistente, nu a clasei de care aparține obiectul. Obiectul devine persistent prin specificarea cuvântului rezervat *persistent,* baza de date care conține obiectele persistente fiind văzută ca un graf de obiecte interconectate prin OID-urile lor. În POET fiecare clasă care se doreşte persistentă, este dotată cu cuvântul persistent care determină acest lucru.

```
Persistent Class freza
{
.....
}
```

În ObjectStore persistența se implementează ca o clasă de memorare. Clientul are acces la mai multe zone de memorare care sunt segmentate, permițînd persistența prin referință. Necesitatea de a specifica clasele ca fiind persistente înseamnă, că atunci când fiecare clasă este definită independent în limbajul de programare al ODBMS-ului, trebuie să existe un mecanism explicit care să transfere acest lucru de la program la OODBMS.

Un alt mod este *accesibilitatea prin căi persistente sau persistența prin moştenire.* În acest mod un obiect este făcut persistent, când este referit de către un alt obiect persistent sau aceasta este o proprietate pe care o moşteneşte de la o clasă persistentă. Toate clasele persistente instanțiază obiecte persistente. Acest lucru este deosebit de avantajos în cazul structurilor complexe. În Versant, persistența se implementează prin intermediul unei librării de clase, care moştenesc aceast concept de la o clasă rădăcină, persistentă. Această clasă se numeşte Pobject.

```
Class freza:public Pobject
{
.....
};
```

Un alt aspect al persistenței pentru un ODBMS, este modul în care o aplicație accesează obiectele persistente, cum sunt aduse obiectele persistente în memorie, cum sunt traversate structurile persistente de către program astfel ca acest lucru să fie transparent şi ce mecanisme sunt necesare pentru a permite accesul. Pot exista două concepte. Unul, permite accesul aplicației la obiectele persistente, citind starea sa, într-un obiect temporar şi apoi dacă obiectul a fost actualizat accesul să se facă în spațiul temporar. După ce actualizarea s-a terminat în spațiul temporar, data temporară este folosită pentru explicitarea

obiectului corespunzător în spațiul persistent. Alt concept este acela prin care pur şi simplu se obțin informații despre obiectul persistent şi se trimit mesaje spre obiectele persistente care solicită operații. Aceste operații sunt realizate în totalitate în spațiul persistent. Datorită acestui fapt accesul la OODBMS înseamnă accesarea obiectului, referindu-l printr-un atribut al unui alt obiect cunoscut de către aplicație. Presupunem că aplicația are un obiect în memorie, care conține o referință la un alt obiect care se află în baza de date. Această situație este prezentată în figura 2.9

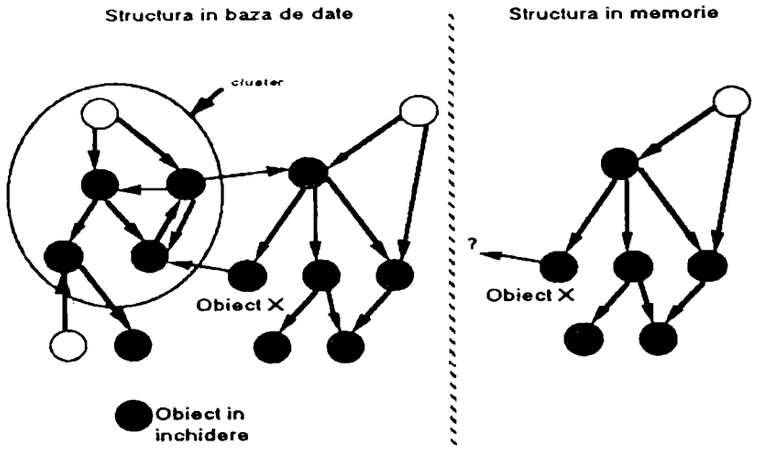


Fig 2.9 Obiecte persistente într-o aplicație

În parte stângă a figurii, sunt reprezentate obiectele din baza de date iar în partea dreaptă sunt reprezentate obiectele aplicației, în memorie. Obiectele încadrate de cerc sunt obiecte accesibile de la obiectul X. Dacă există o referință la un obiect, ODBMS-ul detectează automat dacă obiectul este în memorie sau nu. Dacă nu este în memorie, îl încarcă de pe disc în memoria aplicației. Transparența acestui proces este dată de modul în care pointerii de obiect (OID) sunt implementați în ODBMS şi în aplicațiile ODBMS-ului. Cele mai multe ODBMS-uri folosesc pointeri separați pentru baza de date şi pentru aplicație, legați cu mecanisme de translație a obiectelor din baza de date în memorie. Acest lucru se face pentru că cerințele pentru pointerii din memorie şi cei din baza de date sunt diferite. Tehnica prin care toate referințele din cadrul unui obiect se transformă în pointeri de memorie către locul în care se găsesc acele obiecte în memorie se numeşte *swizzling*. Această tehnică evită prezența tabelelor de descriere a legăturilor. Este indicată folosirea acestei tehnici, dacă sunt accesate un număr mare de obiecte şi aceste sunt menținute permanent în memorie. La scrierea obiectului pe disc, este necesară convertirea pointerilor în OID-uri.

În bazele de date orientate obiect sau în bazele de date relațional obiectuale, persistența este implementată în mod diferit. ROODB-urile asigură persistența obiecteor cu ajutorul tabelelor. O clasă este considerată o tabelă iar fiecare tuplă din relație este considerată un obiect. Obiectele complexe se implementează prin imbricarea tabelelor. În mediul Visual Fox, dialogul EDIT

RELATIONSHIP permite definirea legăturilor persistente între tabele. Tabelele trebuie să existe într-o clasă bază de date.

2.1.8 Integritatea în OODBMS

Acest concept face referire în principal la limbajele de programare orientate obiect, care trebuie să dispună de constructori care să indice restricții de integritate, ce vor testa complectitudinea şi corectitudinea ADT-urilor, adică testează semantica completă a unui tip de date abstract, precum şi corectitudinea sa. Există mai multe căi pentru asigura restricțiile de integritate şi anume:

- Implementarea unor rutine cu restricții în definiția claselor similare cu restricțiile de integritate din bazele de date relaționale. Aceste restricții se regăsesc de fapt asupra obiectelor şi în cazul în care o astfel de restricție este violată sistemul generează erori.
- Introducerea unor restricții asupra variabilelor instanță înainte sau după executarea unor metode

Restricțiile de integrite pot fi impuse asupra operațiilor fundamentale de tipul inserare, actualizare sau ştergere de obiecte. În unele SGBD-uri ca de exemplu Visual Fox sau Oracle pot fi creați *triggeri* care permit impunerea unor restricții pe baza evaluării unor expresii sau a unor programe, care verifică sau impun anumite condiții. La nivelul unor SGBD-uri de tipul Oracle, Visual Dbase, Visual Fox pot acționa restricții de integritate de tipul NOT NULL, UNIQUIE, PRIMARY KEY

2.1.9 Trăsături opționale ale unui OODBS

Aceste trăsături, care vor fi enumerate în continuare, nu sunt întodeauna necesare pentru a defini un sistem de baze de date orientat obiect. Ele sunt următoarele:

- *moştenirea multiplă*, defineşte capacitatea obiectelor de a avea proprietăți de la mai multe clase. Sunt mai multe soluții pentru implementarea acestui concept, pentru a nu intra în conflict cu alte concepte specifice bazelor de date orientate obiect;
- *modelul de verificare,* presupune verificarea sistemului în faza de compilare, astfel încât, soluția optimă este, ca un sistem compilat, să nu mai genereze erori în faza de execuție;
- *modelul de inferență,* presupune că numai modelele de bază declarate în sistem implică existența modelelor temporare;
- *distribuția* este o caracteristică ortogonală a unui sistem orientat obiect, din această cauză, un sistem bază de date poate fi distribuit sau nu;

2.2 Clase de obiecte

Obiectele care au caracteristici comune sunt grupate în clase. O clasă conține descrierea unei structuri de date, metode pentru implementarea detaliilor obiectelor clasei. De aceea, obiectele din aceeaşi clasă partajează aceeaşi structură şi răspund la aceleaşi mesaje. Fiecare obiect al clasei este o instanță a clasei sau o instanță obiect (figura 2.10).

Clasa conține descrierea metodelor care definesc comportamentul obiectelor clasei.

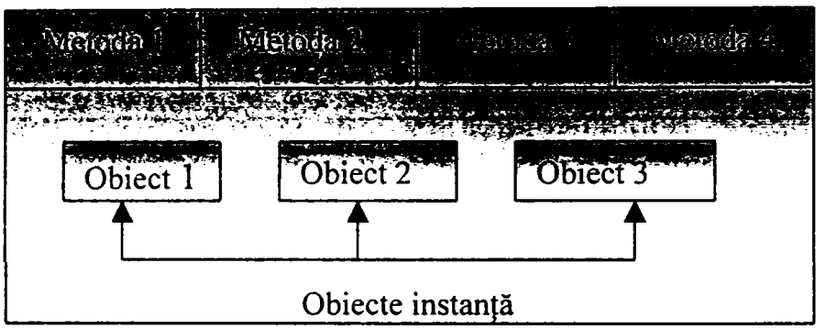


Fig. 2.10 Clasă de obiecte

Considerând exemplul anterior, putem defini o clasă denumită "Freza", pentru a memora obiectele "Freze". Toate obiectele care aparțin clasei, răspund la aceleaşi mesaje şi partajează aceeaşi structură (figura 2.11).

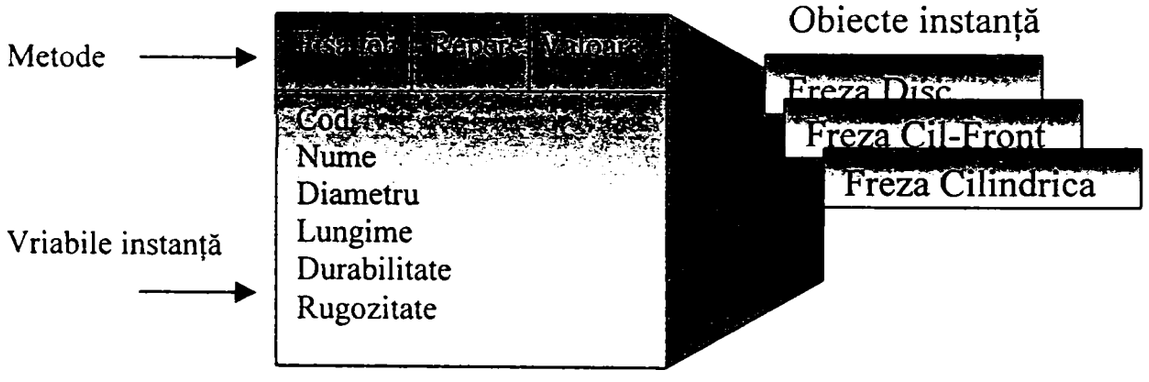


Fig 2.11 Obiecte într-o clasă

Modurile în care sunt definite clasele şi obiectele claselor este diferit, funcție de OODB folosit. În general clasele pot fi definite interactiv prin diverse mecanisme ale mediului sau prin codul program al limbajului specific. De exemplu în mediul CA_Jasmine, clasele şi obiectele ca de altfel orice aplicație poate fi realizată fie cu Jasmine Studio (figura 2.12), interactiv, fie cu limbajul ODQL. Folosind Jasmine Studio, cu ajutorul lui Application Manager, se pot crea sau modifica, clase şi obiecte (butonul New Object).

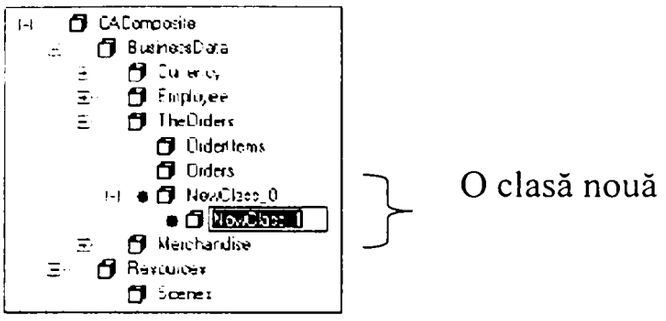


Fig 2.12 Jasmine Studio

În următorul exemplu se prezintă modul de definire a unei clase, a obiectelor şi a metodelor acesteia în Jasmine.

```
defaultCF demoCF;
defineClass Freza
super: ApplicationObject
{
maxInstanceSize: 4;
class:
Integer Nr_freza default: 0;
Freza find( String nume ); instance:
Integer cod_freza unique:;
String nume:;
Bag<String> clasafreze
default: Bag{};
Location atelier;
String diametru;
String lungime;
Integer nr_dinti;
mediaCF::Bitmap photograph;
Void allocatePersonnelNo();
Bag<instance>:
Boolean coLocated();
};
```

Definirea claselor într-un mediu obiectual relațional se face asemănător, fie interactiv fie folosind cod program. În Visual Fox manipulare claselor şi a obiectelor se poate face interactiv folosind Class Browser-ul (figura 2.13).

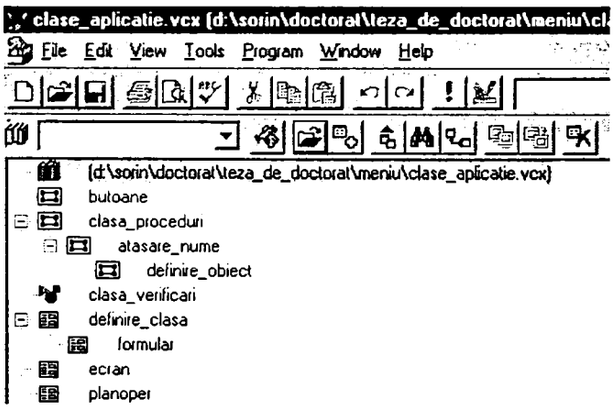


Fig 2.13 Class Browser pentru Visual Fox

Codul program pentru definirea clasei "clasa_proceduri" din figura 2.13 este următorul:

```
DEFINE CLASS clasa_proceduri AS container
        Width = 215
        Height = 58
        *-- numele tabelului care controleaza clasa
        PROTECTED tabelnume
        tabelnume = ""
        Name = "clasa_proceduri"
ADD OBJECT clasa_verificari1 AS clasa_verificari WITH ;
                Top = 12, ;
                Left = 24, ;
                Height = 36, ;
                Width = 96, ;
                Nume = "Clasa_verificari1"

PROCEDURE fabricate
        public varalez,varprodus
        do case
        case alltrim(oform.definire_obiect1.txtden_alezor.value)="Alezoare de mina"
                lode.initialselectedalias="w_de_mana"
                lode.vede()
        case alltrim(oform.definire_obiect1.txtden_alezor.value)="Alezoare de mina pentru alezajele stifturilor
conice 1:50"
                lode.initialselectedalias="w_de_mana1"
                lode.vede()
        case alltrim(oform.definire_obiect1.txtden_alezor.value)="Alezoare cu alezaj conic cu placute din
carburi metalice lipite"
                lode.initialselectedalias="w_alezaj_conic"
                lode.vede()
        endcase
        varalez=cod_alezor
        varprodus=cod_produs
        lode.initialselectedalias="coduri"
        lode.alege()
ENDPROC
ENDDEFINE
```

Definirea clasei, se face cu comanda DEFINE CLASS...ENDDEFINE, definirea unui obiect cu comanda, ADD OBJECT...WITH Proprietăți, iar definirea unei metode, cu comanda DEFINE PROCEDURE... ENDPROC,

2.2.1 Tipuri abstracte de date (ADT)

Un tip de dată descrie obiecte cu caracteristici similare. În limbajele de programare există tipuri predefinite de date (întreg, caracter, logic etc). De asemenea, există constructori de date, de exemplu tipul articol. Ca şi tipurile predefinite de date, tipurile abstracte de date, descriu obiecte asemănătoare. Un tip de dată abstract este diferit de un tip predefinit de dată prin:

1. Operațiile asupra unui tip abstract de dată sunt definite de către utilizator;
2. Tipul abstract de dată nu permite accesul direct la reprezentarea internă a datelor sau la modul de implementare al metodelor.

Cu alte cuvinte, ADT încapsulează definiția sa prin ascunderea caracteristicilor;

Pentru a defini un ADT este necesară definirea:

1. Numelui;
2. Reprezentarea datei sau a variabilei instanță, este necesar să se facă în interiorul unei clase. Fiecare variabilă instanță are un tip de dată care poate fi un tip de bază sau poate fi un alt ADT;
3. Operațiile şi constrângerile ADT-ului sau ale clasei, sunt implementate cu ajutorul metodelor.

În sistemele orientate obiect diferența între clasă şi tip de dată este următoarea: tipul se foloseşte pentru a referi structura clasei şi metodele, iar clasa se foloseşte pentru a referi mai multe obiecte instanță. Tipul de dată este mai mult un concept static, pe când clasa este mai mult un concept dinamic.

Tipul abstract de dată, împreună cu moştenirea permit definirea *obiectelor complexe*. Un *obiect complex* este format prin combinarea altor obiecte într-un set de relații complexe.Un astfel de obiect complex poate fi folosit în sistemul de asigurare a securității bazei de date, folosind diferite tipuri de date, ca de exemplu:

- Tabele;
- Imagini;
- Sunete.

Tipurile abstracte de date definesc atât structura obiectelor cât şi comportamentul acestora. Toate limbajele folosite de către OODBMS-uri folosesc tipuri abstracte de date. Fără acest tip de dată ar fi practic imposibilă realizarea obiectelor complexe.

2.2.2 Identificarea obiectului

Cea mai relevantă parte a unui obiect este identitatea. Ea este reprezentată prin Object ID (OID), care este unică pentru fiecare obiect. OID este asignată de către sistem în momentul în care un obiect este creat şi nu mai poate fi modificată ulterior. În general, nu trebuie să se facă confuzie între cheia primară a unei relații şi OID. Spre deosebire de identitatea obiectului, cheia primară se bazează pe valorile atributelor şi poate fi schimbată oricând, pe când identitatea este asignată de către sistem, nu depinde de nici un atribut al obiectului şi nu poate fi schimbată niciodată. Ea poate fi ştearsă numai dacă obiectul este şters. Mai mult decât atât, OID nu este legată de adresa fizică de pe disk, aşa cum sunt articolele dintr-un model bază de date ierarhică sau în rețea. Această caracteristică permite unui sistem orientat obiect să-şi mențină din punct de vedere fizic independența datelor. Fizic OID-ul este un index într-o tabelă, din care se obține adresa obiectului persistent pe disc, fiind compus dintr-o adresă unică de pagină pe disc şi dintr-un index unic în această pagină.

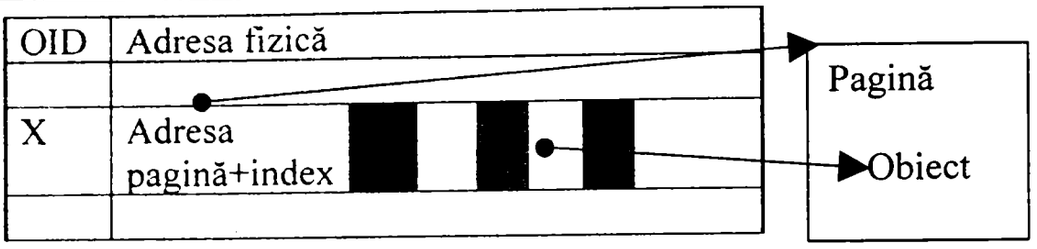


Fig 2.14 Reprezentarea fizică a OID-ului

La nivel de implementare, OID este folosit pentru a realiza legăturile între două sau mai multe obiecte. Putem spune că, un obiect este cunoscut ca o colecție de obiecte. Poate fi subliniat următorul aspect: *Dacă în modelul relațional, atributul poate conține doar o singură valoare, care poate fi folosită pentru legarea diferitelor linii din tabele, în modelul obiectual nu este necesară legarea în acest mod a obiectelor.*

Starea obiectului este un set de valori pe care îl au atributele la un moment dat. Cu alte cuvinte, dacă starea obiectului poate varia, identitatea rămâne aceeași. Dacă vrem să modificăm starea obiectului trebuie să modificăm atributele obiectului. Pentru a schimba valorile atributelor trebuie transmise mesaje obiectului. Aceste mesaje implică existența metodelor.

2.2.3 Comunicarea între obiecte prin mesaje

Pentru a înțelege mesajele şi metodele, să ne imaginăm obiectul ca şi o coajă de nucă. Miezul nucii (nuca) reprezintă structura obiectului, iar învelişul reprezintă metodele (figura 2.15)

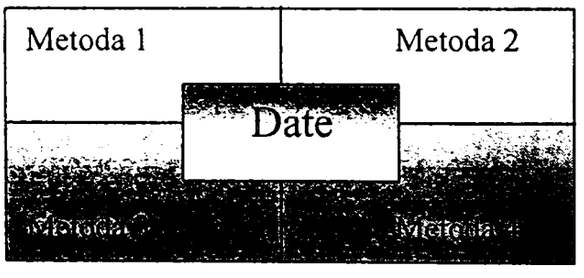


Fig 2.15 Reprezentarea obiectului şi a metodelor

Pentru ca o operație să poată funcționa pe un obiect trebuie ca ea să fie implementată printr-o metodă. Metodele sunt folosite pentru a schimba valorile atributelor obiectelor sau pentru a returna valorile atributelor selectate. Ele reprezintă o acțiune reală, ca de exemplu, schimbarea cursurilor la care participă un student, afişarea numelui unui student etc. Ele sunt echivalente cu procedurile din limbajele de programare. Fiecare metodă este identificată printr-un nume şi are un corp. Corpul metodei este format din instrucțiuni scrise într-

un anumit limbaj de programare. Fizic o metodă poate exista independent, sub forma unui fişier sau poate fi incorporată în baza de date obiectuală.

De exemplu, putem defini metoda Media_Produselor_Fab_An, care returnează media aritmetică a celor două atribute "Media Produselor Fab. pe Sem 1" şi "Media Produselor Fab. pe Sem2

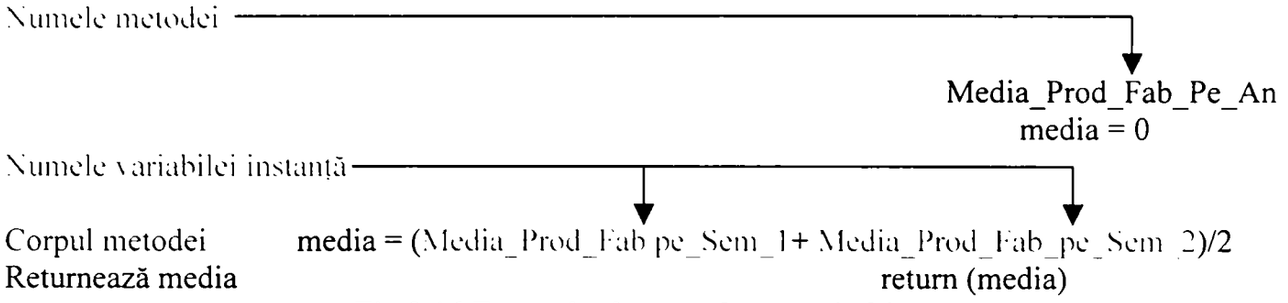


Fig 2.16 Exemplu de metodă a unui obiect

Examinând acest exemplu, putem deduce că, metoda poate accesa variabilele instanță (atributele) obiectului pentru care este definită.

Apelarea unei metode înseamnă transmiterea unui mesaj la un obiect. Mesajul este transmis, specificând obiectul spre care este transmis, numele metodei şi argumente. El este format din selectori şi argumente. Selectorul este de fapt numele metodei care trebuie să se execute la obiectul dorit, iar argumentele, se transmit parametrilor metodei. Structura internă a obiectului nu poate fi accesată direct de mesajul transmis de către alt obiect. Refuzul accesului la structura obiectului asigură integritatea acestuia şi ascunde structura internă a obiectului. Proprietatea de a ascunde detaliile interne ale obiectului a fost definită în capitolul precedent şi se numeşte *încapsulare.* Un obiect poate trimite mesaje pentru a schimba sau a interoga starea altui obiect. Interogarea implică consultarea valorilor variabilelor instanță. Corpul unei metode poate conține referiri la metodele altui obiect, aşa cum se arată în figura 2.17

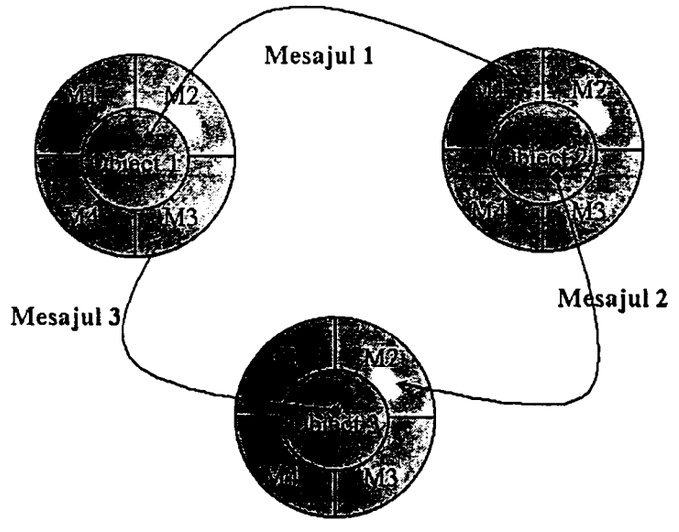


Fig 2.17 Mesaje între obiecte

2.3 Superclase, Subclase şi Moştenirea în sistemele baze de date orientate obiect

Clasele sunt organizate în ierarhii de clase, dacă fiecare clasă are o singură clasă părinte şi într-o reţea de clase, dacă o clasă poate avea mai multe clase părinte. Generalizarea este folosită pentru a clasifica obiectele din clase, care-şi partajează resursele comune. De exemplu, generalizarea clasei "Scule de găurit" figura 10, cuprinde alezoare şi burghie. Din figura 2.18, se poate observa că "De mână", "De maşină", "De maşină c. cil", sunt subclase ale alezoare, care la rândul ei, este o subclasă a clasei "Scule de găurit". Clasa scule de găurit poate fi considerată ca o superclasă pentru clasa burghie.

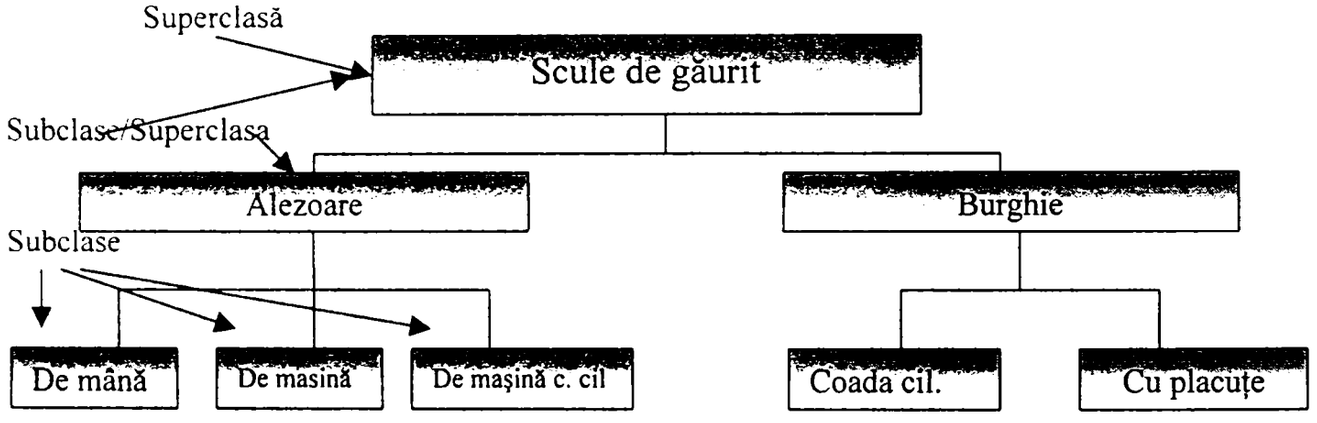


Fig. 2.18 Moştenirea simplă

Ierarhia de clase introduce un concept orientat obiect deosebit de puternic, numit *moştenire*. Aceasta este însuşirea firească a unui obiect din ierarhia de clase, de a moşteni structura datelor şi metodele clasei de care aparţine. De exemplu, obiectul De mână moşteneşte structura datelor şi comportamentul (metodele) de la clasele:alezoare şi scule de găurit. Este important de subliniat că, în mediile orientate obiect, se poate folosi cod reutilizabil. Există două tipuri de moştenire, şi anume : *moştenirea simplă şi moştenirea multiplă.*

Moştenirea simplă, există când o anumită clasă are numai o singură superclasă. Cele mai multe sisteme orientate obiect suportă moştenirea simplă.

Când sistemul transmite un mesaj la un obiect instanţă, este parcursă ierarhia de clase pentru a fi găsită metoda potrivită, folosind următorii paşi:

1. Sunt scanate clasele la care aparţine obiectul;
2. Dacă metoda nu este găsită, atunci sunt scanate superclasele.

Procesul de scanare se repetă până când metoda este găsită sau până este parcursă întreaga ierarhie de clase. În cazul în care metoda nu este găsită se semnalează eroare.

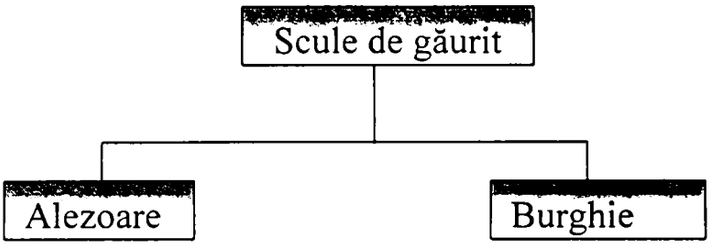


Fig 2.19 Exemplu de ierarhie de clase

De exemplu, într-o ierarhie de clase, ca în figura 2.19, lansarea mesajului de elaborare al "Fişei tehnologice" caută metoda cu acest nume în ierarhia de clase. Această metodă se aplică tuturor obiectelor din clasă.

Moştenirea în cazul OODB, prezintă două aspecte şi anume *moştenirea instanţelor* şi *moştenirea metodelor*. Un prim caz este acela în care se moştenesc toate instanţele din superclase în subclase şi se permite metodelor din subclase să apeleze instanţele superclasei, fără nici o restricţie, lucru specific în Smalltalk şi GemStone. Această înseamnă că este violată proprietatea de încapsulare obiectului. Pentru a păstra conceptele obiectuale, unele medii soft, prevăd posibilitatea de a defini obiectul instanţă: public, privat sau vizibil la nivelul subclasei. Acest lucru este implementat în Visual Dbase, Visual Fox, prin cuvintele rezervate Protect, Protected, Public. O instanţă declarată publică este vizibilă de toate entităţile din sistem. O instanţă declarată privată (protected) atunci nici o entitate nu o poate apela. Dacă instanţa este vizibilă la nivel de subclasă, aceasta înseamnă că ea poate fi apelată de către toate metodele subclaselor acelei clase. În mod asemănător o subclasă poate moştenii metodele unei superclase.

Moştenirea multiplă (figura 2.20), există în cazul în care o clasă are ca şi părinţi mai multe superclase. Aşa cum s-a arătat anterior această caracteristică este opţională în OODBMS.

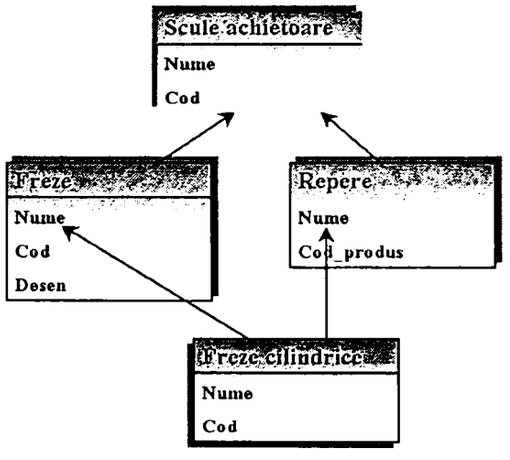


Fig 2.20 Moştenirea multiplă

Clasa respectivă moşteneşte caracteristicile superclaselor. De asemenea, ea moşteneşte toate metodele superclaselor. În cazul în care există variabile instanță sau metode cu acelaşi nume este necesar ca sistemul orientat obiect să decidă care variabilă sau metodă este folosită. Există trei posibilități de rezolvare a unei astfel de probleme:

- Emiterea unui mesaj de eroare într-o fereastră în care se explică problema care apare;
- Declanşarea unei întrebări către utilizator, pentru a corecta o anumită valoare sau pentru a întreprinde o acțiune care să conducă la rezultatul dorit;
- Afişarea unui rezultat nesemnificativ.

Conflictele care pot să apară datorită moştenirii multiple sunt rezolvate de către sistemele orientate obiect prin implementarea unor rutine care să trateze moştenirea multiplă referitoare la subclase.

Fiind mai greu de implementat o serie de limbaje nu o suportă. Ea este implementată în C++ dar nu este implementată în Java, Delphi, Visual Fox.

Aşa cum s-a arătat în capitolul precedent, *polimorfismul* şi *suprascrierea,* sunt caracteristici ale programării orientate obiect. Legat de folosirea claselor şi a metodelor, suprascrierea (overloading), este folosită de foarte multe ori. Pentru clasa "Scule de găurit " putem considera o metodă care calculează prețul unui produs.

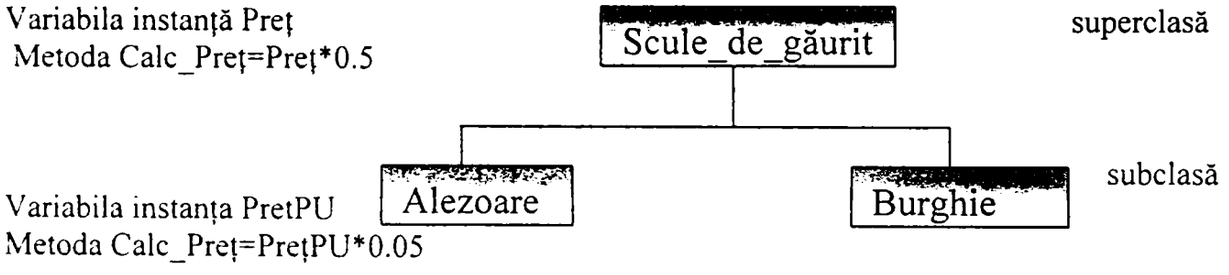


Fig. 2.21 Suprascrierea

Dar aceast preț este diferit funcție de categoria de scule, una pentru subclasa "Alezoare" şi alta pentru subclasa "Burghie". Astfel, am suprascris metoda "Calc_Preț", existentă la nivelul superclasei "Scule de găurit", la nivelul subclasei figura 2.21.

Polimorfismul permite ca obiecte diferite să răspundă în moduri diferite la acelaşi mesaj. El este o trăsătură foarte importantă a sistemelor orientate obiect, pentru că permite obiectelor să se comporte funcție de caracteristicile lor proprii. În termenii sistemelor orientate obiect, polimorfismul înseamnă că:

1. Putem folosi acelaşi nume pentru o metodă definită în mai multe clase ale ierarhiei de clase;

2. Utilizatorul poate trimite același mesaj, adică apelează același nume

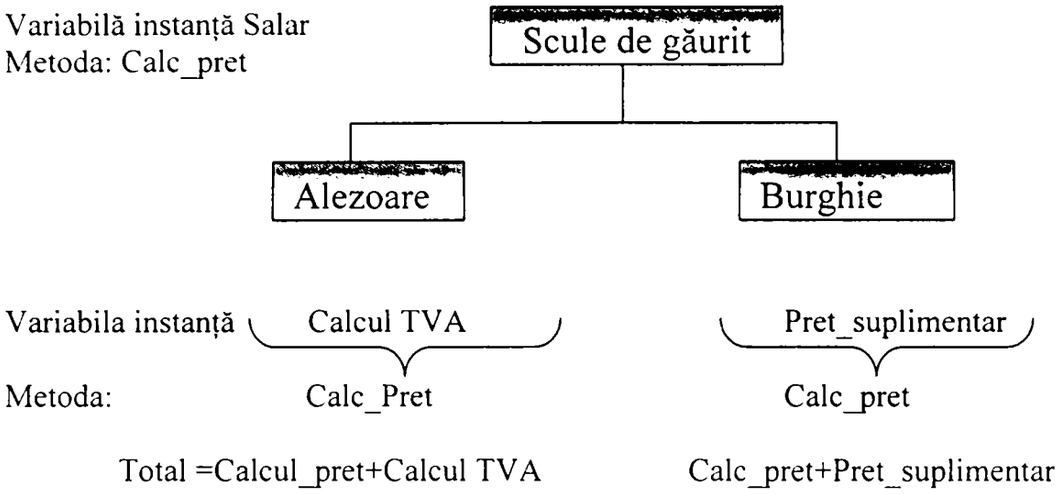


Fig. 2.22 Polimorfismul

de metodă cu aceeași parametrii, la diferite obiecte care aparțin la clase diferite, ele generând un răspuns corect.

Polimorfismul prezentat în figura 2.22 arată că:

1. Calc_pret pentru clasa "Alezoare" este o metodă rescrisă din metoda cu același nume care aparține superclasei Scule de găurit ;
2. Metoda Calc_pret definită în superclasa "Scule de găurit" este refolosită de către clasele "Alezoare" și "Burghie"

În concluzie, polimorfismul permite folosirea codului reutilizabil în proiectarea aplicațiilor orientate obiect. El se găsește în multe OODBMS sau RODBMS ca de exempluVersan, Poet, GemStone, Delphi, Visual Dbase, VisualFox

2.4 Scheme obiectuale

2.4.1 Diagrame de obiecte

Conform [Khosh 93] și [Hamme 83], un sistem orientat obiect poate fi reprezentat grafic. Un obiect este reprezentat printr-un dreptunghi, în care este introdus numele variabilei instanță. Considerăm clasa "Burghie" și obiectele corespunzătoare (figura 2.23).

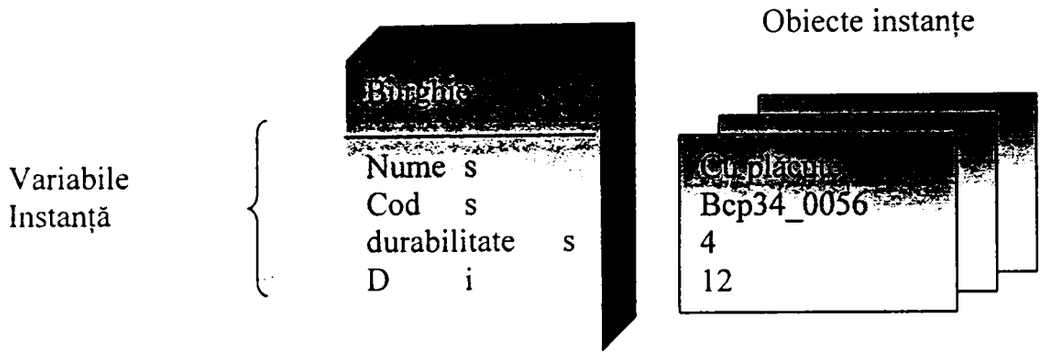


Fig. 2.23 Clasa "Burghie" şi obiectele corespunzătoare

Starea obiectului Burghie este reprezentată în figura 2.24.

Burghie	
Nume	Cu placuțe
Co	Bcp3 _0056
durabilitate	4
D	12

Fig 2.24 Starea obiectului

Variabila instanță "durabilitate" poate fi privită ca un atribut derivat. Atributele derivate pot fi implementate prin metode. Astfel, poate fi creată metoda calc_durabilitate pentru clasa Burghie. Această metodă poate returna data de la care scula respectivă nu mai poate fi folosită la un număr de folosiri prestabilit. Metodele sunt folosite datorită avantajelor pe care le prezintă în încapsularea şi moştenirea claselor.

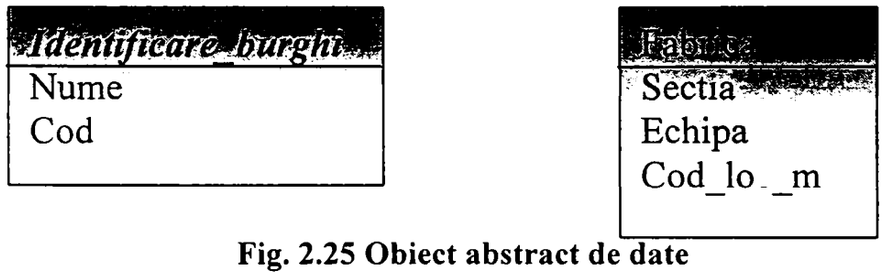


Fig. 2.25 Obiect abstract de date

Evenimentele orientate obiect permit crearea tipurilor abstracte de date din tipurile de bază. De exemplu, Identificare_burghiu şi Fabrica sunt atribute compuse, care pot fi implementate ca ADT, figura 2.25. Rezultă că, clasa "Burghie" conține atribute care fac referință la obiecte ale altor clase sau la tipuri abstracte de date.

Schema obiectuală este echivalentă cu schema bazei de date. Schema obiectuală este folosită pentru a reprezenta starea obiectelor. Diagrama obiectuală pentru clasa "Burghie" este reprezentată în figura 2.26.

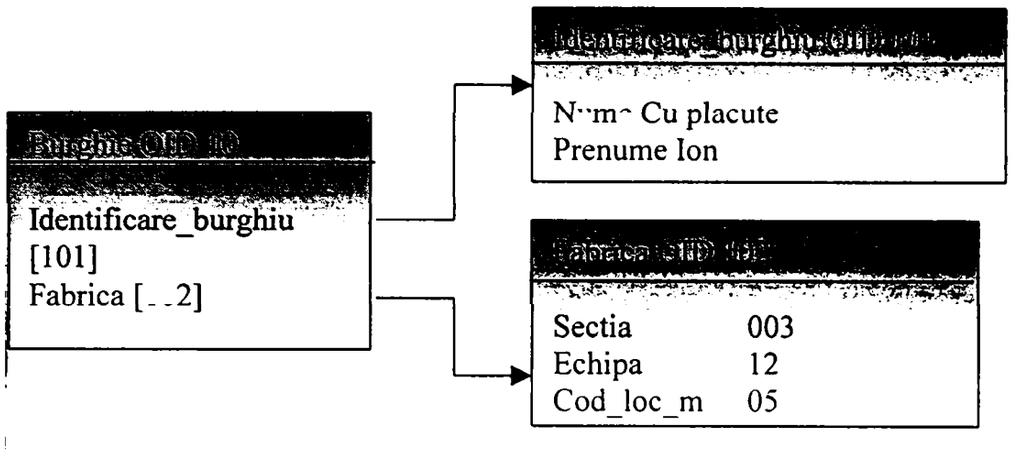


Fig. 2.26 Diagrama obiectuală pentru clasa "Burghie"

Atributele: Identificare_burghiu şi Fabrica, conţin acum câte un număr de identificare al instanţei din clasă sau ADT, în locul valorii de bază. În general, numărul de identificare este independent de starea obiectului.

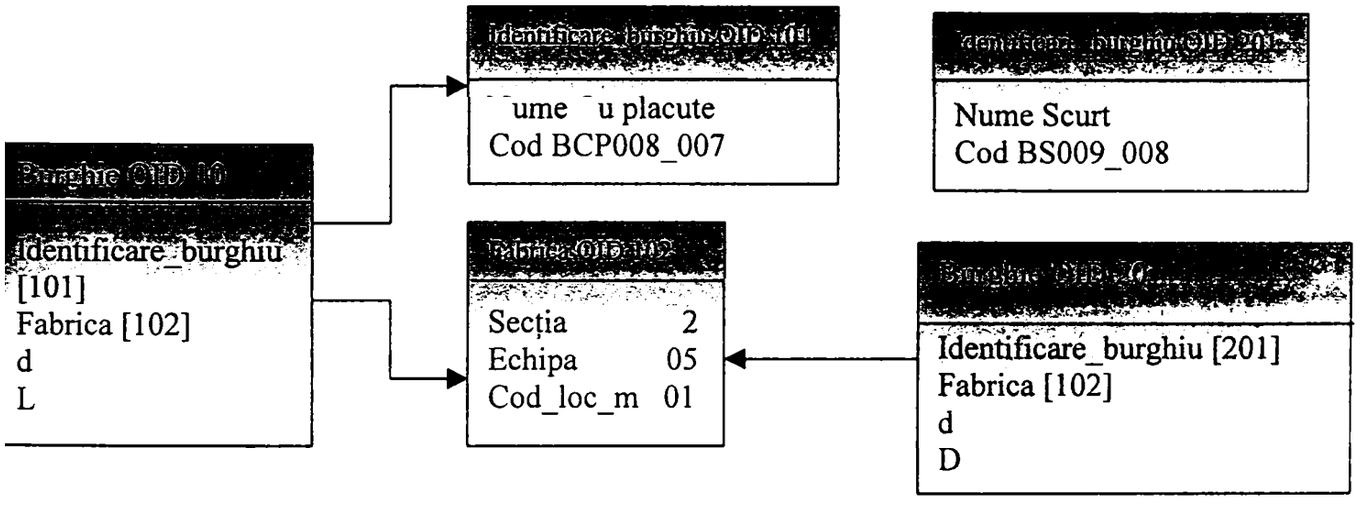


Fig. 2.27 Referinţa partajată a obiectelor

Referinţa partajată a obiectelor, poate să apară în cazul a două obiecte ale aceleaşi clase, care fac referinţă la acelaşi obiect instanţă, figura 2.27.

2.4.2 Clase şi subclase în cadrul schemei obiectuale

În general, o subclasă moşteneşte proprietăţile unei clase. Această proprietate permite folosirea unei etichete pentru a descrie legătura între clase, în interiorul unei ierarhii. Astfel, Freze este o Sculă aşchietoare, la fel şi Cuţite este o Sculă aşchietoare(figura 2.28).

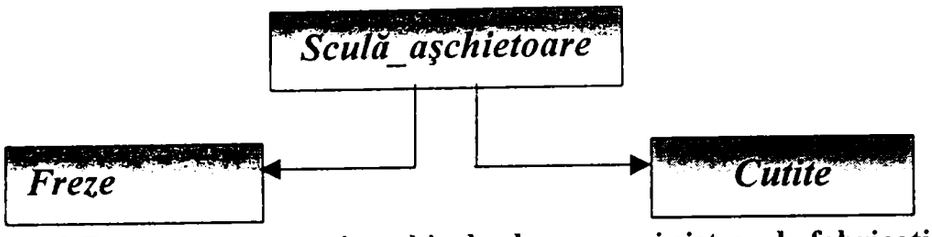


Fig 2.28 Exemplu pentru o ierarhie de clase a unui sistem de fabricaţie

Fiecare subclasă, moşteneşte proprietăţile superclasei şi poate avea proprietăţi în plus (figura 2.29).

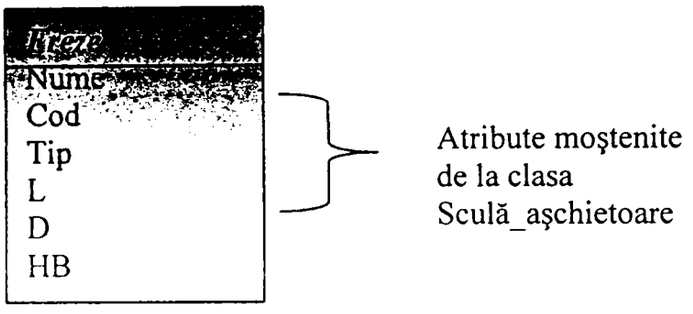


Fig. 2.29 Atributele clasei "Freze"

Relaţia dintre clasă şi superclasă este de tipul 1:1(one to one).

Conform lui [Baner 87] regulile care trebuie respectate atunci când se modifică schema externă pentru ca aceasta să nu devină inconsistentă se împart în patru categorii:

- Rezolvarea conflictelor cauzate de moştenirea multiplă precum şi redefinirea atributelor şi metodelor dintr-o clasă. În această categorie sunt cuprinse următoarele reguli:
  1. *regula de precedenţă a subclaselor faţă de clase* prin care dacă un atribut sau metodă a unei clase este definit cu acelaşi nume ca un alt atribut sau metodă al unui obiect dintr-o superclasă atunci definiţia specificată în subclasă are întâietate faţă de definiţia specificată în superclasă;
  2. *regula de precedenţă între superclase* cu origini diferite arată că dacă diverse superclase au proprietăţi sau metode cu acelaşi nume, dar cu origini diferite, atunci numele proprietăţii sau metodei primei superclase este moştenit de către subclasă.
  3. *regula de precedenţă între superclase cu aceeaşi origine* arată că dacă mai multe proprietăţi sau metode au acelaşi nume şi aceeaşi origine atunci ele sunt moştenite o singură dată.
- Propagarea modificărilor la subclase
  1. *regula pentru propagarea modificărilor* afirmă că în cazul în care o proprietate sau o metodă se modifică într-o clasă acest lucru se propagă în subclase cu excepţia acelor clase sau subclase în care atributul sau metoda au fost redefinite;

2. *regula de propagare a modificărilor în cazul conflictelor* arată că modificările se propagă doar la acele clase la care nu apar conflicte de nume;
3. *regula pentru modificarea domeniilor* afirmă că domeniul unui atribut poate fi modificat doar folosind generalizarea

- Reguli privind modificarea sau ştergerea relaţiilor de moştenire dintre clase
  1. *regula de inserare a superclaselor* arată că dacă o clasă este adăugată pe lista de superclase a unei alte clase ea devine ultima dintre superclasele acelei clase;
  2. *regula de eliminare a superclaselor* arată că dacă o superclasă a unei clase este eliminată atunci clasa devine subclasă pentru toate superclasele clasei eliminate;
  3. *regula de inserare a unei clase* într-o schemă arată că dacă ea nu are specificată nici o superclasă ea devine subclasă a clasei rădăcină;
  4. *regula de eliminare a unei clase* se realizează prin aplicarea succesivă a regulei 2.
- Reguli privind manipularea obiectelor compuse

2.4.3 Relaţii interclase.

2.4.3.1 Legături clasă atribut

O relaţie între clase este creată când un atribut este definit printr-o referinţă la un ADT. Relaţia interclase [Rob 95] , este diferită de relaţia existentă între clasă şi subclasă.

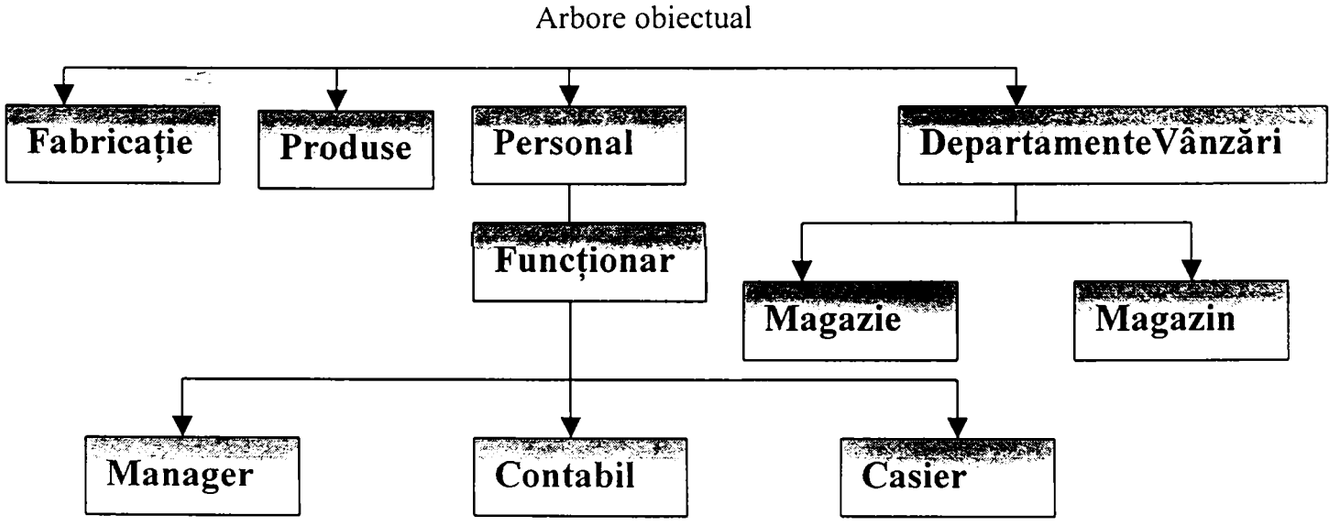


Fig 2.30 Structura obiectuală aunei societăţi economice

Considerăm structura unei societăți economice, din punct de vedere obiectual, figura 2.30.

Ierarhia de clase prezentată în figură este folosită ulterior pentru a prezenta relațiile de tip 1:M şi M:N şi pentru a demonstra implementarea relațiilor de tip many to many.

2.4.3.2 Reprezentarea relațiilor de tip 1:M

Bazat pe ierarhia de clase prezentate, se poate considera că, fiecare angajat lucrează într-un singur departament vânzări şi fiecare departament vânzări conține mai mulți angajați. Figura 2.31 reprezintă modul în care poate fi implementat un astfel de tip de relație:

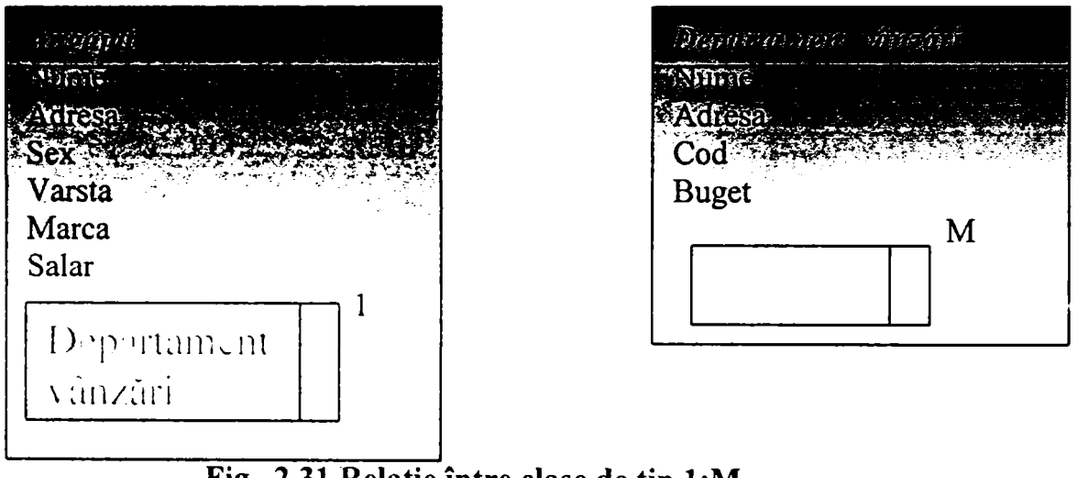


Fig. 2.31 Relație între clase de tip 1:M

Din figura 2.31, se observă că obiectul Departament vânzări este inclus în obiectul Angajat şi invers. În general, pentru reprezentarea relațiilor dintre clase trebuie respectate următoarele reguli:

- Clasele care aparțin relației sunt încadrate în dreptunghiuri pentru a le scoate în evidență cât mai mult;
- Linia dublă în partea dreaptă a dreptunghiului semnifică faptul ca relația este obligatorie;
- Conectivitatea este reprezentată prin eticheta ataşată fiecărui dreptunghi. În acest caz punem 1 în dreapta Departament vânzări a obiectului Angajat pentru a indica că fiecare angajat lucrează într-un singur departament. Litera M care apare în obiectul Departament vânzări semnifică faptul că fiecare departament de vânzări are mai mulți angajați.

Pentru a semnifica tipul de relație şi tipul de conectivitate păstrăm notația de la diagrama E-R. În general relația de tip 1:M este reprezentată în ambele clase, acest lucru permițând inversarea relației, dacă este cazul.

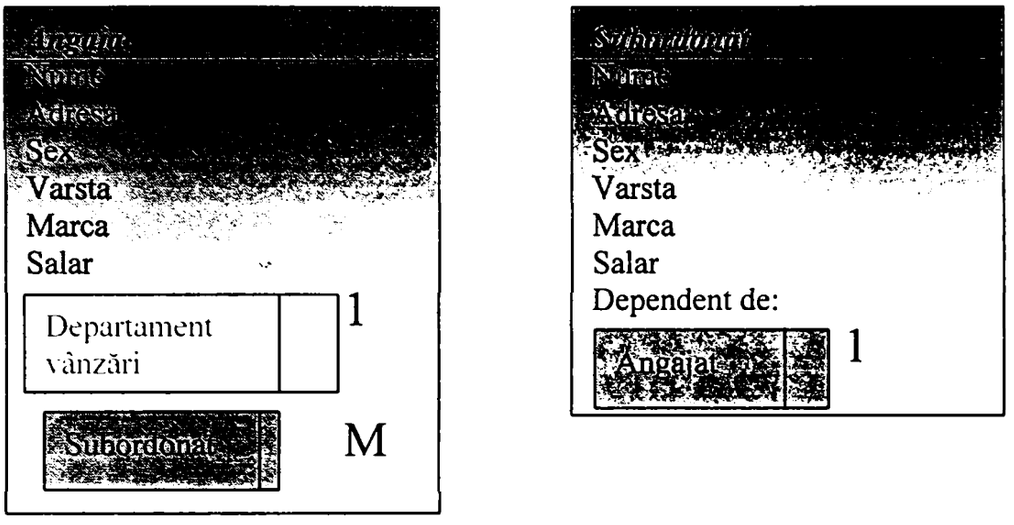


Fig. 2.32 Exemplu de relație între clase de tip 1:M

Un alt exemplu de relație de tip 1:M poate fi ilustrat considerând angajatul şi subordonații săi. Considerăm în primul rând clasa Subordonați ca subclasă a clasei Personal. Putem considera că fiecare manager este un angajat, fiecare angajat este o persoană şi fiecare subordonat este o persoană, dar nu fiecare subordonat este un angajat. Clasa Subordonat este opțională în relația cu Clasa Angajat şi ea are o relație de tipul 1:M cu clasa Angajat. Totodată clasa Angajat este obligatorie pentru clasa Subordonat (figura 2.32).

2.4.3.3 Reprezentarea relației de tip M:N

Se consideră clasele Fabricant şi Produse (figura 2.33). O relație de tipul M:N

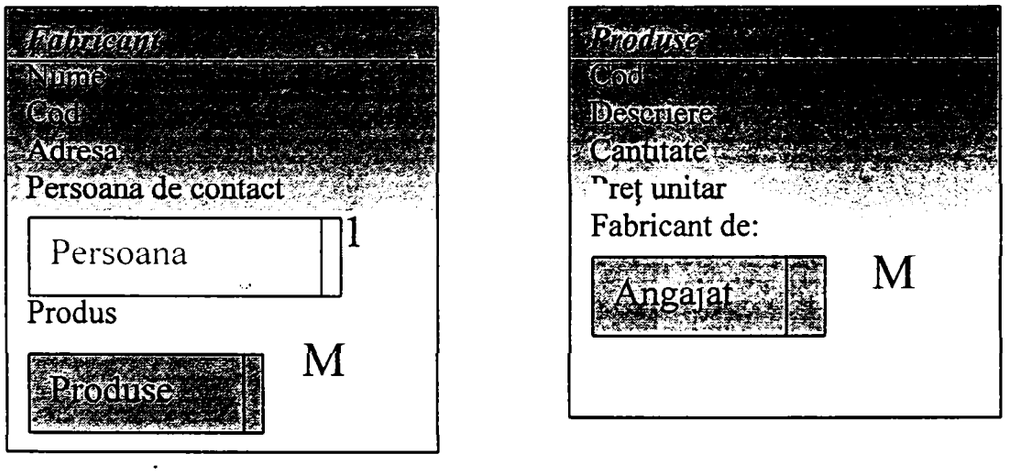


Fig 2.33 Relație între clase de tipul M:N

există între Fabricant şi Produse. Fiecare produs poate fi fabricat de către mai mulți fabricanți şi fiecare fabricant poate produce mai multe produse. Atributul

Persoana de contact referă o instanță din clasa Persoana. Relațiile de tip M:N pot fi re_rezentate ca o intersecție de clase.

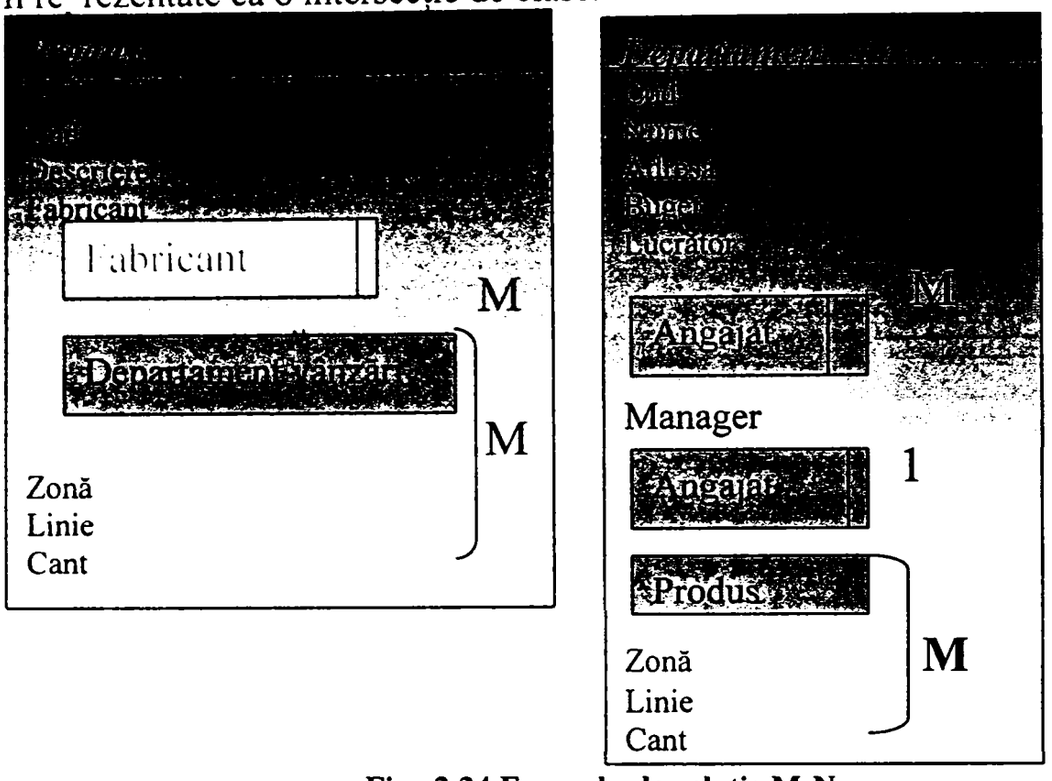


Fig. 2.34 Exemplu de relație M:N

Presupunem că adăugăm noi condiții într-o relație M:N care ne permit să urmărim date suplimentare pentru fiecare pereche de obiecte instanțe. De exemplu, expandăm relația între Produse și Departament vânzări în sensul în care Departamentul vânzări poate conține mai multe produse și fiecare produs poate fi localizat în mai multe departamente de vânzări. În plus, vrem să urmărim cantitatea și localizarea fiecărui produs și fiecare departament. Ilustrăm acest lucru în figura 2.35.

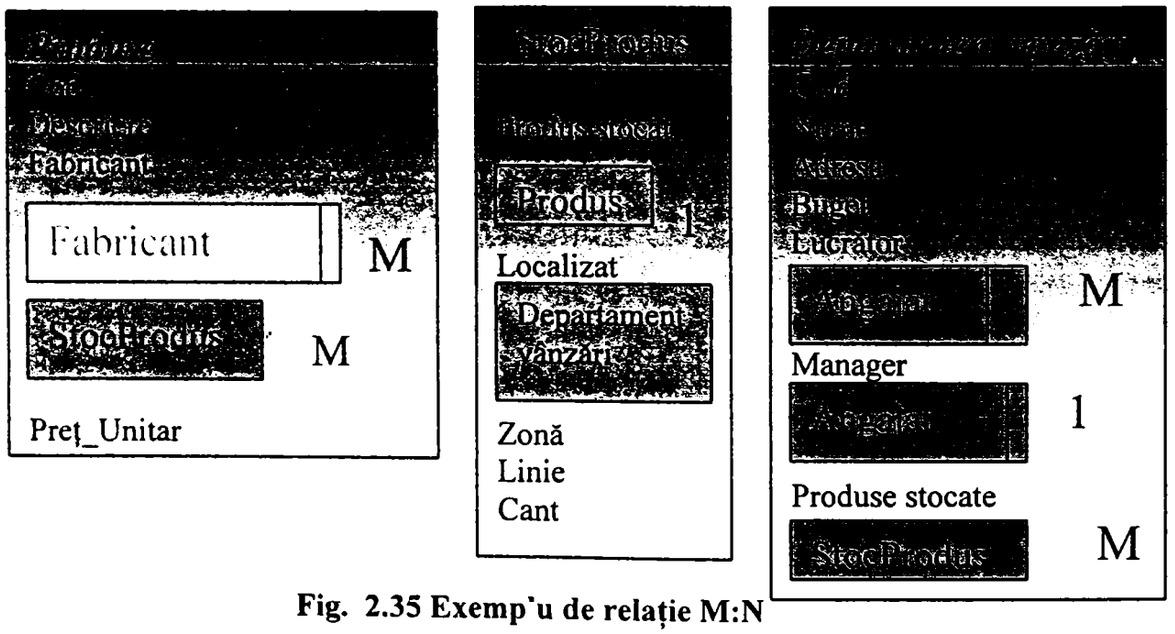


Fig. 2.35 Exemp'u de relație M:N

Paranteza dreaptă este folosită în figura 2.34 pentru a indica că atributele incluse sunt tratate ca o unitate logică. De aceea, fiecare instanță Produs poate conține mai multe apariții Departament vânzări, fiecare acompaniată de valorile corespunzătoare pentru atributele Zonă, Linie sau Cant. Din punct de vedere al unei relații M:N, putem defini o clasă intersecție care să fie conectată atât cu clasa Departament vânzări cât şi cu clasa Produse. În acest scop, creem clasa StocProdus care conține instanțe obiect ale claselor Produs şi Departament vânzări precum şi valori pentru fiecare dintre atributele Zonă, Linie şi Cant.

Spațiul obiect poate fi reprezentat ca în figura 2.36. Acesta conține câteva puncte important de analizat din punct de vedere al proiectării bazei de date obiectuale:

- Instanțele clasei StocProdus conțin referințe la câte o instanță a fiecărei clase specificate: Produs şi Departament vânzări. Clasa StocProdus, ca intersecție de clase, este necesară numai în cazul în care, ulterior, este definită informația suplimentară;
- Instanța obiect Produse conține în schema obiectuală o colecție de obiecte ale clasei StocProdus, fiecare conținând şi instanțe ale obiectului Departament vânzări. Instanța obiect Departament vânzări conține în schema obiectuală, o colecție de obiecte ale clasei StocProdus, fiecare conținând şi instanțe ale obiectului Produs. Rezultă că cele două relații reprezintă două aplicații diferite ale aceluiaşi obiect;
- Referințele interclase folosesc identificatori de obiecte (OID) pentru a referi obiecte în scopul introducerii lor în spațiul obiect;
- Valoarea dintre parantezele drepte reprezintă OID-ul unei instanțe ale aceleaşi clase. Colecția de clase reprezintă o clasă de obiecte în care fiecare obiect conține o colecție de obiecte de aceeaşi clasă;
- În modelul obiectual nu este necesar să folosim operatorul Join pentru a combina datele din diferite obiecte pentru că, un anumit obiect, poate conține referințe la alte obiecte. Aceste referințe sunt automat activate când obiectul este accesat.

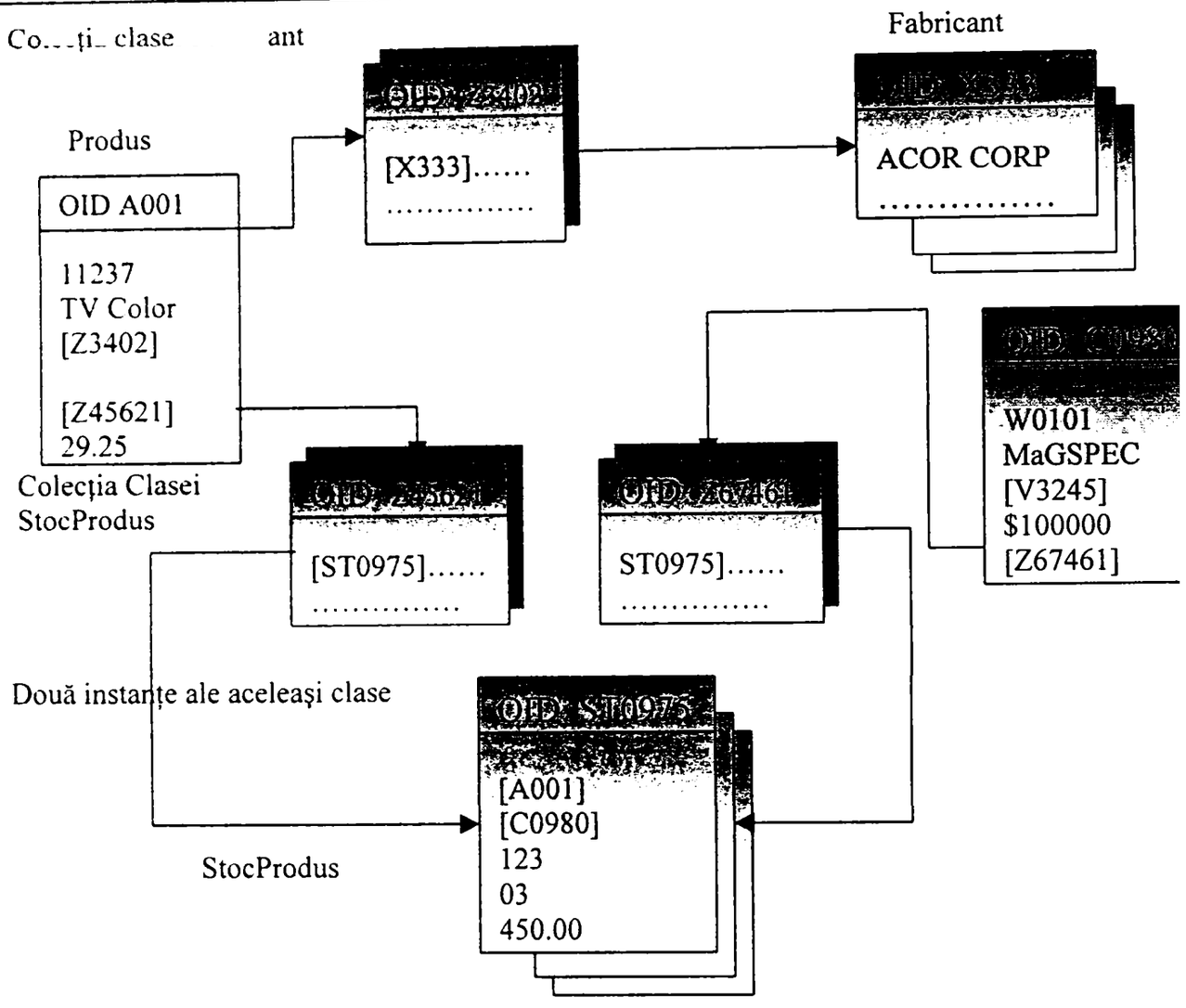


Fig. 2.36 Spaţiul obiect

2.5 Legare întârziată şi legare timpurie

O caracteristică importantă a unui OODBMS este aceea ca unele atribute ale obiectelor, să conţină obiecte care definesc într-un anumit moment un anumit tip de dată, care ulterior se poate schimba. Astfel un obiect poate conţine o variabilă numerică pentru o variabilă instanţă dată şi următorul obiect poate conţine o valoare şir de caractere, pentru aceeaşi variabilă instanţă. Această proprietate se numeşte legare întârziată. Cu alte cuvinte, tipul de dată al unui atribut nu este cunoscut decât în momentul execuţiei obiectului. Datorită acestui fapt, două obiecte diferite din aceeaşi clasă, pot conţine valori de tip diferit pentru acelaşi atribut. În proiectarea bazelor de date relaţionale, tipul atributului este definit în momentul în care structura relaţiei este definită. Acest concept de definire al tipului de dată, este cunoscut sub numele de *legare timpurie*. Legarea timpurie permite bazei de date, de a găsi tipuri de date, pentru fiecare atribut, în procesul de compilare sau în procesul de definire. Prin acest concept, un atribut odată definit prin atribuirea unui tip de dată, rămâne permanent legat de acest tip

de dată. Modelul orientat obiect al bazelor de date permite definirea ADT. În exemplul următor, datele abstracte Tip_inventar, Furnizor, Greutate sau Bani sunt asociate cu variabile instanță în momentul definirii. Datorită acestui fapt, proiectantul poate defini operațiile cerute pentru fiecare tip de dată. De exemplu, tipul de dată Greutate, poate avea metode care prezintă greutatea unui produs în diferite unități de măsură. La fel Prețul unui produs poate fi prezentat în diverse monezi.

În conceptul de *legare târzie sau legrea dinamică*, tipul de dată al atributului nu este cunoscut decât în momentul în care el este folosit. De aceea, un atribut poate avea orice tip de valoare asignată. Legarea târzie are importanță asupra polimorfismului care permite obiectului să stabilească ce metodă devine activă în momentul în care este activ în execuție. În tabelul 1 sunt prezentare diferențele dintre modelul relațional și cele două concepte ale modelului obiectual.

Tip relațional					
Tabelă	Inventar		Legarea timpurie		
ibut	Nume	Tip de data	Clasa		Inventar
	Tip_inventar	Numeric	Variabile instanță	Nume	Tip

În *legarea dinamică* (dynamic binding), sistemul leagă în momentul execuției selectorii mesajelor de metodele vizate. Acest lucru este important din punct de vedere al polimorfismului, deoarece același mesaj (nume de metodă) poate fi utilizat astfel pentru clase diferite și clasa de obiecte, nu este cunoscută până în momentul execuției.

OODBMS este o tehnologie nouă, care permite managementul datelor complexe specifice aplicațiilor moderne de proiectare CAD/CAM, ingineria software, procesarea informațiilor geografice etc, mult mai bine decât în cazul folosirii tehnologiei relaționale.

2.6 Arhitectura ODBMS-urilor

OODBMS-urile au în general o arhitectură total diferită de bazele de date relaționale. Natura acestei diferențe este prezentată în figura 2.37.

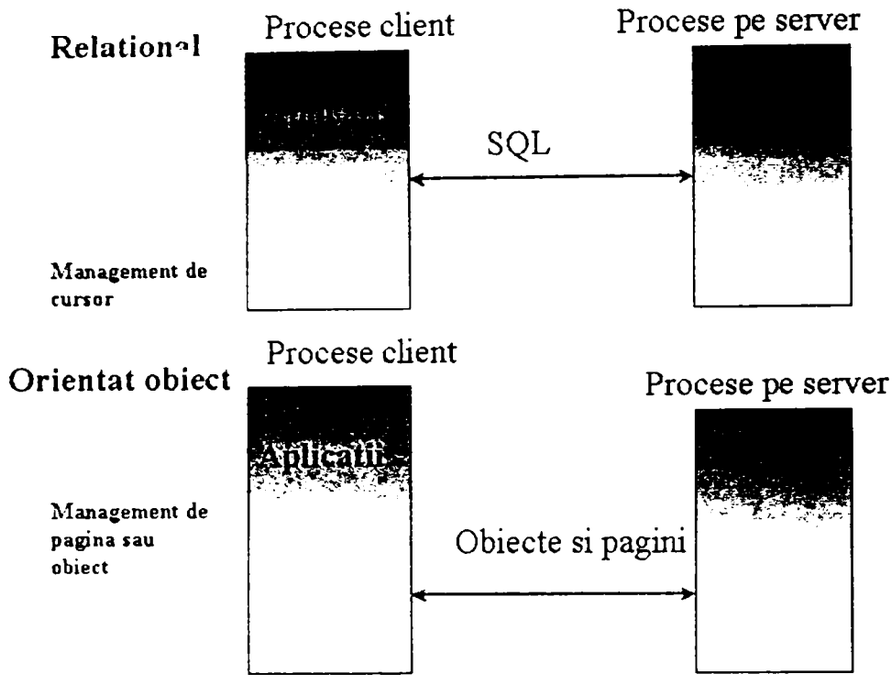


Fig 2.37 Arhitecturi RDBMS şi OODBMS

Atât RDBMS cât şi OODBMS-ul au arhitecturi client server dar diferența esențială constă în modul în care se împart sarcinile între client şi server. În sistemele relaționale arhitectura este proiectată ca aplicațiile să trimită interogări spre server. Toate procesele de interogare se fac în server care returnează rezultatele ce satisfac clientul De asemenea serverul lansează tranzacții recuperări de informații etc. Responsabilitatea clientului este de a conduce aplicația şi de a controla rezultatele returnate de către server. Această arhitectură evită traficul de rețea pentru că sunt returnate numai rezultatele cerute de către aplicație. Ea este o soluție bună în cazul în care serverul este mult mai puternic decât clientul. În cazul OODBMS-ului rolul clientului devine mai important. Aceste arhitecturi se bazează pe conceptul date transportate (*data shipping*). În acest concept, datele sunt transportate de la server la client astfel încât multe dintre procesele OODBMS-ului sunt executate la client. Acest lucru are două avantaje:

1. În cazul structurilor complexe de obiecte, cerința este ca aplicația să urmărească referințele de la un obiect la altul. Conceptul dată transportabilă mută închiderea datelor la aplicație şi suportă o bună legătură între aplicație şi accesul la următorul obiect persistent cerut de către aceasta;
2. Funcțiile DBMS-ului sunt transferate de la server la client. Aceasta permite serverului să facă legătura cu mai mulți clienți şi să folosească la maximum resursele clienților legați în rețea

Diferențele de arhitectură crează un mare avantaj pentru aplicațiile orientate obiect în OODBMS-uri față de RDBMS. Conform conceptului *date transportabile* obiectele sunt memorate pe disc în unități specifice numite

pagini. De aceea ODBMS trebuie să asigure memorarea obiectelor în pagină şi legătura dintre obiecte şi pagină. Modul în care se face legătura dintre client şi server poate fi *server pagină*, unde clientul şi serverul interacționează prin pagină sau grupuri de pagini şi *server obiect* unde clientul şi serverul interacționează ca obiecte individuale.

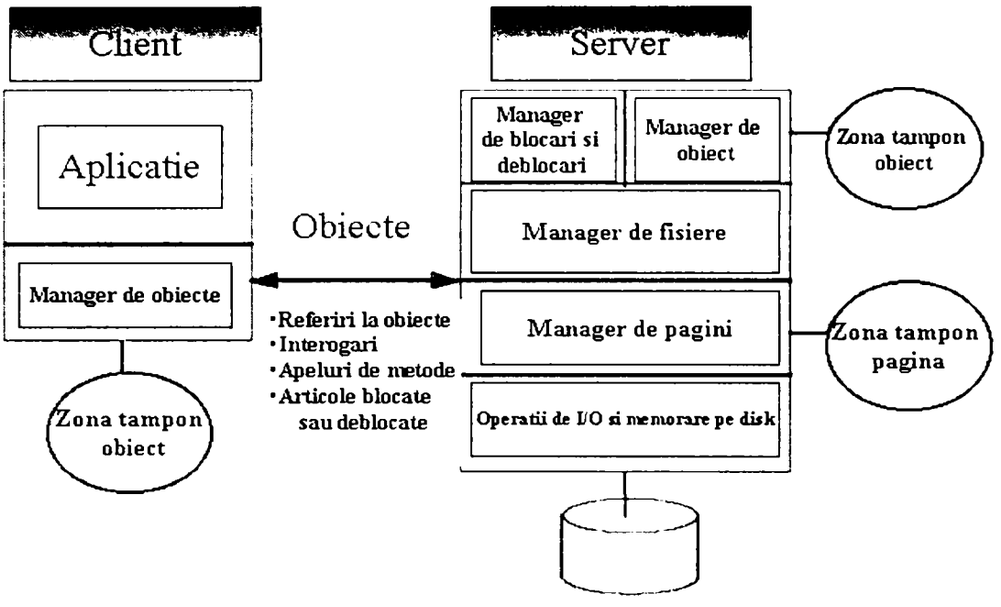


Fig 2.38 Arhitectură de tip server obiect pentru ODBMS

Figura 2.38 arată o configurare posibilă pentru *serverul obiect*. În acest tip de arhitectură unitatea de bază pentru transfer între server şi client este obiectul, cu toate că şi alte tipuri de date pot fi transferate. Aceasta înseamnă că între client şi server trebuie să existe o componentă, care poate manipula obiecte individuale. Cu alte cuvinte, între client şi server există zone tampon în care se găsesc obiectele accesate cel mai recent. Când clientul are nevoie de un obiect, îl va căuta prima dată în zona tampon (cache). Dacă el nu este găsit se transmite o cerere către server. Acesta îl va căuta în zona sa tampon. Dacă nu-l va găsi, atunci căutarea se va face în paginile recent accesate. Dacă este găsit el este transmis clientului. Altfel serverul caută obiectul pe disc şi-l va returna clientului. Arhitectura obiect server are o serie de avantaje:

1. O metodă care apelează un număr relativ mic de date dintr-o structură complexă, poate rula pe server fără a mai fi necesară transferarea unui volum mare de date le client pentru procesare;
2. Serverul cunoaşte cu exactitate, care obiecte au fost accesate de fiecare aplicație, asigurând astfel controlul concurenței;
3. Numai obiectele necesare aplicației sunt transferate, riscul ca ele să fie eronate scăzând considerabil

Această arhitectură are şi o serie de dezavantaje. În cel mai rău caz trebuie să existe o procedură separată de accesare a obiectelor pentru fiecare obiect în parte. Alt dezavantaj este acela că arhitectura complică proiectarea serverului Arhitectura *server pagină* este prezentată în figura 2.39.

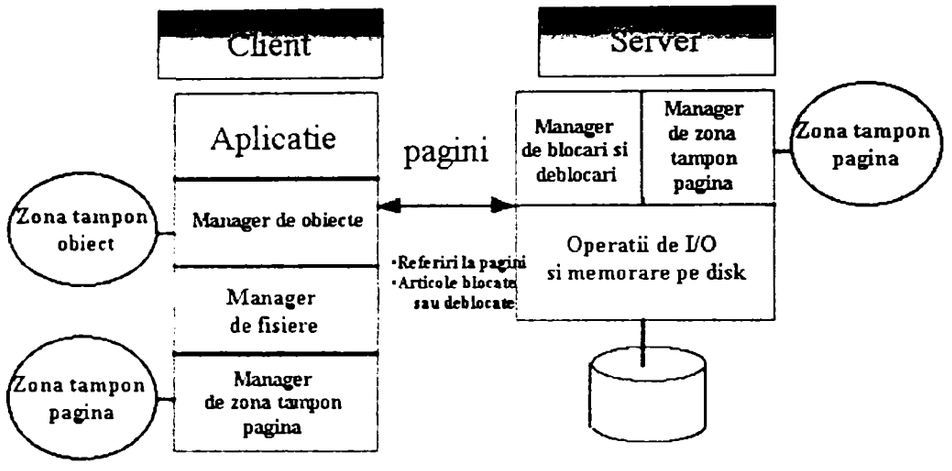


Fig 2.39 Arhitectură de tip server pagină pentru ODBMS

În arhitectura *server pagină* unitatea de bază pentru transfer este pagina. Soft-ul ODBMS rulează în întregime pe unitatea client şi unitatea de transfer între client şi server este pagina sau grupul de pagini. La fel ca şi la celălalt tip de arhitectură şi în acest caz există avantaje şi dezavantaje. Principalul avantaj constă în faptul că, plasează majoritatea proceselor sistemului pe staţia client. Rolul serverului este astfel minimizat. Un dezavantaj al acestui tip de arhitectură constă în faptul că metodele pot fi evaluate numai de către clienţi. Alt dezavantaj constă în faptul că nivelele de control ale concurenţei obiectelor sunt greu de implementat.

Datorită avantajelor şi dezavantajelor pe care le prezintă cele două tipuri de arhitecturi s-a ales arhitectura *server fisier*, care este un hibrid între cele două tipuri de arhitectură. Ea foloseşte serviciul de citire şi scriere la distanţă, al fişierelor de la Sun (figura 2.40).

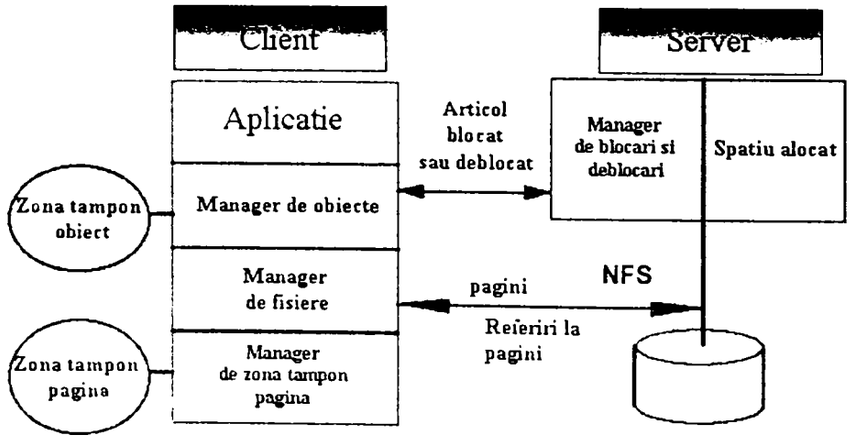


Fig 2.40 Arhitectură de tip server fişier pentru ODBMS

Avantajul acestui tip de arhitectură constă în faptul că NFS rulează în modul kernel folosind pentru citire şi scriere pagini evitând nivelul utilizator care micşorează performanţele. Pe de altă parte dezavantajul constă în faptul că scriere în NFS este mai înceată.

2.7 Tipuri de ODBMS

Comportamentul obiectului, înseamnă execuţia operaţiilor asupra obiectului, adică modul în care se execută metodele asupra obiectului. Acest concept este arătat în figura 2.41.

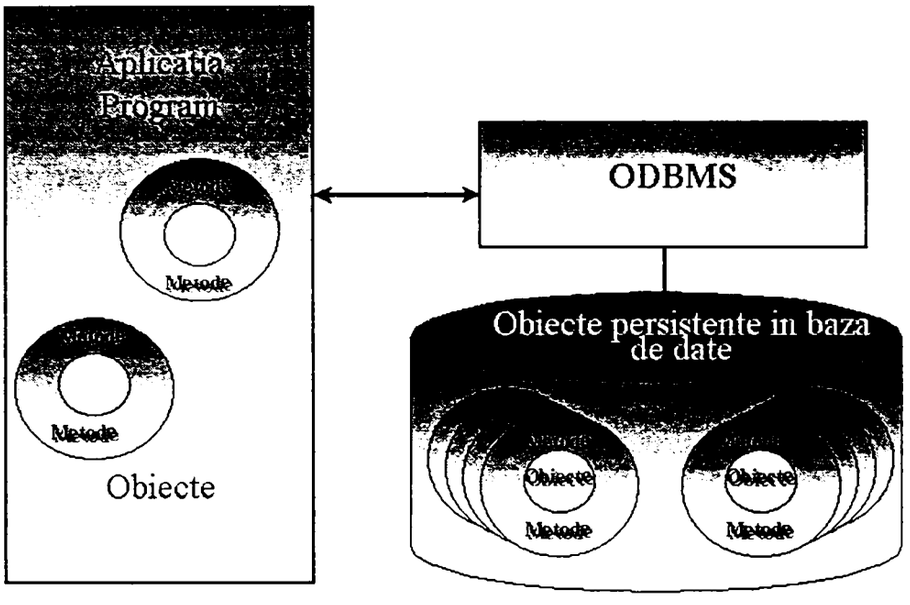


Fig 2.41 Tipuri de ODBMS-uri

În multe ODBMS obiectele sunt definite ca şi clase, în limbajul de programare aferent şi definiţiile sunt înregistrate ca şi tipuri persistente în baza de date. Metodele asociate cu aceste clase sunt compilate o dată cu aplicaţia şi sunt memorate în fişiere binare. Obiectul bază de date conţine starea obiectului, în forma unei structuri de date persistente, care este folosită pentru a defini complet obiectele în memorie, în momentul în care obiectele sunt accesate de către aplicaţie. Acest lucru este transparent, aplicaţia comportându-se ca şi cum obiectele se găsesc memorate în totalitate în baza de date. Acest tip de OODBMS poartă numele de *ODBMS pasiv* pentru că ea însăşi nu poate executa o metodă oarecare decât dacă obiectul este mutat în zona de lucru a aplicaţei înainte de execuţia metodei (figura 2.42).

ODBMS-urile care pot stoca în totalitate obiectele se numesc *ODBMS-uri active* (figura 2.43). Aceste tipuri de ODBMS-uri sunt cele mai uzuale. Ele

permit implementarea constrângerilor de integritate şi menţinerea tuturor

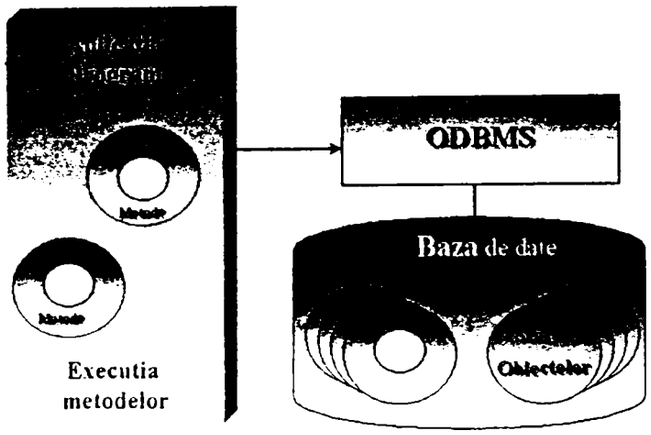


Fig 2.42 ODBMS pasiv

semanticilor în cadrul aplicaţiilor. Acest lucru poate fi făcut şi cu OODBMS-urile pasive dar de multe ori aceasta presupune soft suplimentar. ODBMS-urile active, permit o mai mare flexibilitate a obiectelor, în cadrul reţelelor de calculatoare.

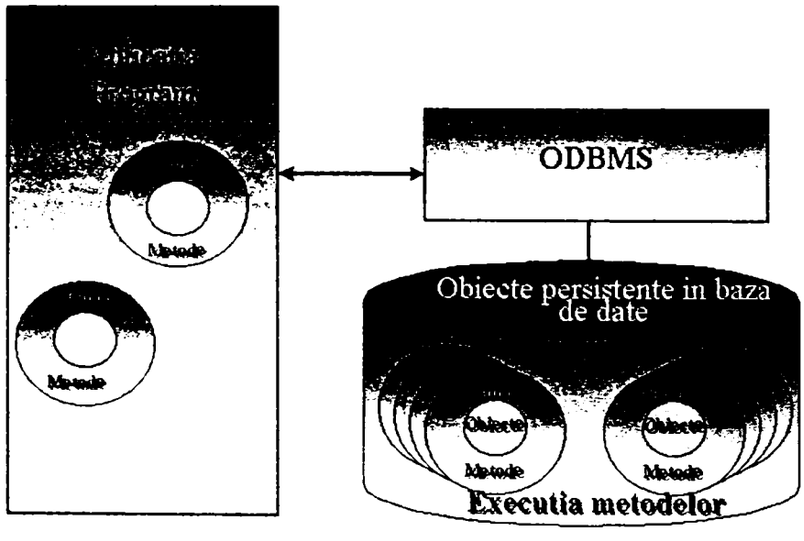


Fig 2.43 ODBMS activ

De multe ori conceptul prin care un OODBMS este activ sau pasiv este confundat cu conceptul arhitectural, prin care un ODBMS este un server de obiecte sau un server de pagini.

O modalitate de a folosi un ODBMS pasiv, ca şi unul activ, este acela de folosire a conceptului de *client partajat* (figura 2.44). Combinaţia dintre acestea două, oferă ODBMS-ului facilităţile unuia activ. Astfel, ODBMS pasiv, este folosit pentru execuţia unor aplicaţii speciale, a căror rol este de a implementa obiecte, cu un comportament controlat centralizat, după care este accesat clientul partajat de către aplicaţiile convenţionale.

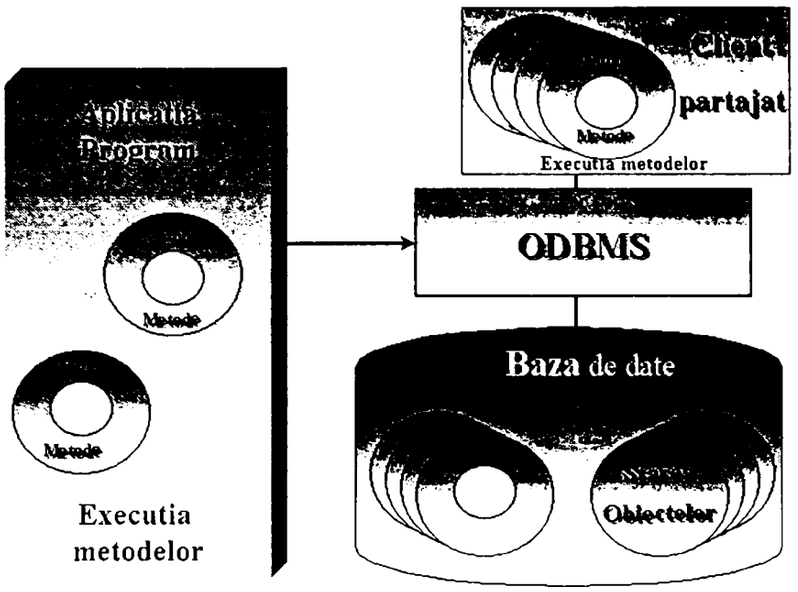


Fig 2.44 ODBMS client partajat

2.8 Standardizări ale OODBMS

2.8.1 Standardul ODMG

ODMG-93 este un set de standarde pentru OODBMS propus de către Object Data Management Group (ODMG). Acesta este un consorțiu de producători şi vânzători de OODBMS, ca de exemplu POET, Object Design, O2, Versant etc, care au încercat impunerea unor standarde valabile, în lumea bazelor de date orientate obiect. El conține:

- un model obiectual comun;
- un limbaj de definire obiectual (ODL). Acesta corespunde limbajului de definire al datelor comun sistemelor de baze de date;
- un limbaj de interogare obiectual (OQL) folosit pentru interogări şi pentru actualizarea obiectelor. Acesta se bazează în mare măsură pe limbajul SQL static, având diverse caracteristici funcție de mediul obiectual în care este folosit;
- limbaje de programare orientate obiect asigurând:
  - cuplarea la limbajul C++;
  - cuplarea la limbajul Smalltalk;
  - cuplarea la limbajul Java.

Modelul obiectual trebuie să respecte conceptele obiectuale, unitatea fundamentală fiind obiectul. Obiectele sunt clasificate pe tipuri. Toate obiectele de acelaşi tip au caracteristici comune. Comportarea obiectului este definită de către o mulțime de operații pe care le poate executa obiectul de un anumit tip. Fiecare operație, specifică pentru un anumit tip de obiecte, are un nume,

argument şi tip. Starea obiectului este definită de proprietăţile sale. Proprietăţile sunt valorile atributelor sale sau legăturile dintre obiect şi alte obiecte. Atributele sunt o categorie de proprietăţi definite pentru un singur tip de obiect. Relaţiile sunt o categorie de proprietăţi definite între două tipuri de obiecte. Ca şi atributele ele nu sunt obiecte. Ele pot fi de tipul unu la unu, unu la mai multe, mai multe la mai multe.

Limbajul de definire al obiectului (ODL), poate fi o extensie a unui limbaj de programare independent. El poate împrumuta sintaxa unui astfel de limbaj, de exemplu C++ sau Java.

Limbajul de definire al interogărilor (OQL). Folosind ODL, OQL furnizează diverse facilităţi pentru interogările specificate. Implementarea limbajului obiectual de interogare este diferită funcţie de OODBMS-ul cu care se lucrează. De exemplu limbajul ODQl din Jasmine.

Limbajele de legătură pot fi în general limbaje independente (C++, Smalltalk etc)

Modelul obiectual ODMG prevede o mulţime bogată de constructori pentru definirea obiectelor. Obiectele pot fi atomice sau structurate. Astfel obiectele pot fi de anumit tip dar pot fi şi tabele. OQL prevăzut de acest tip de standardizare este capabil de a realiza interogări cu orice tip de obiecte. Datorită unor neclarităţi existente în acest tip de standardizare sau făcut anumite modificări şi revizuiri aupra lui ODMG-93, ajungându-se la ODMG-95. Principalele critici care s-au adus sistemului de standardizare ODMG-93 au fost:

1. Imposibilitatea creării unor metode complexe în cazul datelor imbricate;
2. OQL permite interogări pe un singur tip de obiecte;
3. Sintaxă diferită faţă de SQL clasic în cazul în care semantica era de fapt aceeaşi;
4. Omiterea unor concepte majore ale SQL ca de exemplu vederi, triggeri etc;
5. Nu există implementate comenzi de inserare, ştergere, adăugare de obiecte;
6. Permite definirea numelor multiple pentru obiecte, specificarea duratei de existenţă a unui obiect etc.

Importante îmbunătăţiri au fost aduse odată cu publicarea [Catte 00] şi [Conno 01]. ODMG 2.0 a fost complet revăzut faţă de precedentele. El reprezintă un consens pentru componenta tehnologică a produselor şi limbajelor, permiţând portabilitatea pe diferite sisteme. Rolul acestui standard este de a combina limbajele de programare şi DBMS-urile într-un singur concept, în scopul îmbunătăţirii performanţelor acestora. El este un ghid important în acţiunile ce vor fi întreprinse, în bazele de date orientate obiect. ODMG2.0 reprezintă pentru bazele de date orientate obiect ceea ce SQL reprezintă pentru bazele de date relaţionale. Îmbunătăţirile majore ale standardului publicat în 1997, cuprind:

- O nouă legătură de cuplare pentru limbajul Java;
- O versiune complet revizuită a modelului de obiecte, cu un metamodel nou care susține semantica bazelor de date obiectuale pentru multe limbaje de programare;
- Un standard pentru forma exterioară a datelor şi a schemei de date, care permite schimbul de date între bazele de date.

ODMG 3.0 a apărut în ianuarie 2000. El conține o serie de îmbunătățiri referitoare la interfața Java. Unele dintre aceste îmbunătățiri au fost făcute cu scopul de a folosi RDBMS-urile cu tehnologia relațional obiectuală. Cu alte cuvinte se poate folosi standardul orientat obiect pentru bazele de date, in vederea im lementării unui model relațional

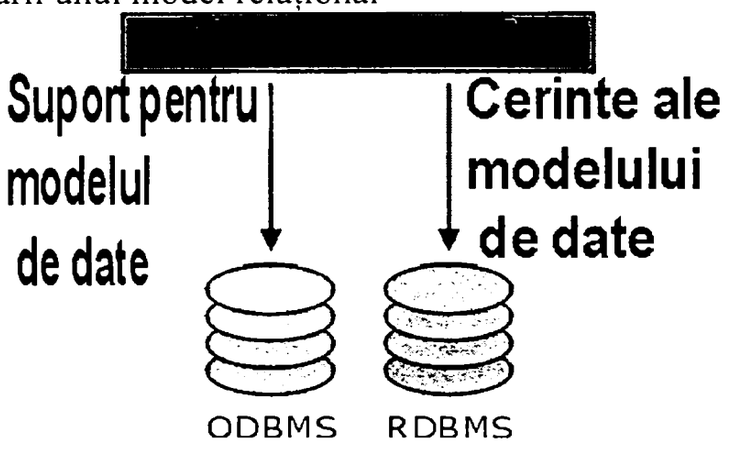


Fig 2.45 Standardul ODMG

Standardul ODMG are drept scop asigurarea transparenței interfețelor şi de a realiza aplicațiile realizate pe tehnologiile relaționale şi obiectuale cât mai portabile.

Proprietățile de bază pentru modelare, care stau la baza standardului ODMG 3.0 sunt *obiectul* şi *literalul*. Diferența între aceste două concepte se bazează pe conceptul *identificator de obiect*. Obiectele au un singur identificator, aşa cum s-a arătat, pe când literalii pot avea mai mulți identificatori. Atât obiectele cât şi literalii sunt impărțiți în mai multe categorii de către *tip*. Obiectele de acelaşi tip, pot avea acelaşi comportament definit de către mulțimea metodelor ataşate, şi aceeaşi stare definită de către proprietățile lor. Proprietățile pot fi atribute sau legături cu alte obiecte, iar comportamentul este definit de către un set de operații pe care le execută obiectul sau care sunt executate pe obiect. În standardul ODMG 3.0, modelul obiect permite definirea unui singur tip de operație a cărei nume trebuie să fie unic. Totodată standardul ODMG 3.0 face diferență între *comportamentul moştenit* al obiectului şi moştenirea *stări şi a comportamentului*. Relația de tip *is_a* se bazează pe comportamentul moştenit al obiectului, care permite stabilirea unor relații de moştenire între obiecte de acelaşi tip. Relația de tip *extends* defineşte moştenirea stării şi a comportamentului între obiecte de acelaşi tip. Ea defineşte totodată o

relație de moștenire între clase. Baza de date este definită ca și o mulțime de instanțe ale unui tip extins. Schema Bazei de date ODMG 3.0 este definită ca și o ierarhie de clase unde interfețele pot fi definite să specifice, comportamentul comun al claselor. Interfețele pot fi legate prin relații de tip *is_a*, pe când clasele pot fi legate doar prin relații de tip *extends*. Relația existentă între clasă și interfață este de tip *is_a*. Datorită acestui fapt în schemele standardului ODMG, clasele și interfețele coexistă.

2.8.2 SQL3

ANSI și ISO au adus modificări standardului SQL [ISO 98], [Amble 00-1] pentru a putea suporta managementul datelor orientate obiect. Specificațiile SQL3 permit folosirea ADT-urilor incluzând metode, identificatori de obiecte subtipuri, polimorfismul și integrarea cu limbaje externe. Totodată SQL3 permite definirea tabelelor incluzând definiții de tuple și identificatori de tuple. El are inclus un mecanism pentru folosirea moștenirii. SQL 3 prevede două căi pentru folosirea modelului obiectual și anume tuplele în tabele (relații) și instanțele ADT. Și acest mod de standardizare a generat o serie de critici care se referă la:

1. SQL3 necesită o singură ierarhie, pentru definirea ierarhiei de tabele. Astfel dacă tabela C moștenește proprietăți de la tabelele A și B, tabelele A și B trebuie să fie descendenții unei tabele comune D;
2. O interogare nu poate folosi o singură tabelă din structura ierarhică de tabele;
3. Nu este posibil să se determine tabela în care se găsește o anumită linie (tuplă);
4. Liniile nu au identitate, ele au nevoie de cheie primară;
5. Modelul obiectual general nu definește moștenirea pentru ADT-uri

Unele dintre aceste critici sunt îndreptățite, dar căile în care au fost prezentate pot cauza confuzii.

2.8.3 Standardul OMG

The Object Management Group (OMG), [OMG 92], [Darwe 98] este un consorțiu industrial a cărui scop este dezvoltarea unei arhitecturi orientate obiect pentru bazele de date distribuite. Aceștia pun accentul pe obiective ca interoperabilitate, refolosire, portabilitatea aplicațiilor. Standardele OMG au fost publicate în câteva documente:

- Object Management Architecture (OMA) Guide;
- Common Object Request Broker Architecture (CORBA);
- Common Object Services Specification (COSS), Volume I.

OMA identifică şi caracterizează componentele, interfețele şi protocoalele care apar în arhitectura distribuită(figura 2.46).

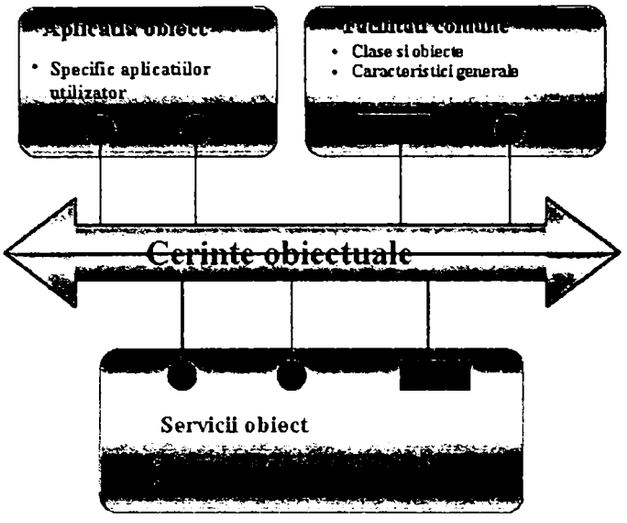


Fig 2.46 Standardul OMA

Părțile componente ale lui OMA sunt:

- Object Request Broker (ORB) permite obiectelor să lanseze şi să recepționeze cereri şi răspunsuri în starea distribiută.
- Serviciile obiectuale (Object Services) sunt o colecție de servicii care definesc funcțiile de bază pentru folosirea şi implementarea obiectelor;
- Facilitățile comune (Common Facilities) este o colecție de servicii care pot fi aplicate în diverse aplicații;
- Aplicații obiectuale (Application Objects) sunt obiecte specifice unei anumite aplicații utilizator.

În OMA obiectul este identificabil, este o entitate încapsulată, care are prevăzute o serie de servicii cerute de către client. Acesta poate fi un alt obiect sau o aplicație oarecare. Operația înseamnă un serviciu care poate fi cerut. Operația are o semnătură care descrie valorile parametrilor şi rezultatele returnate. Componenta ORB (figura 2.47), suportă clienții care execută cereri asupra obiectelor. Cererea este o operație asupra unui obiect care poate avea unul sau mai mulți parametrii sau poate să nu aibă nici un parametru.

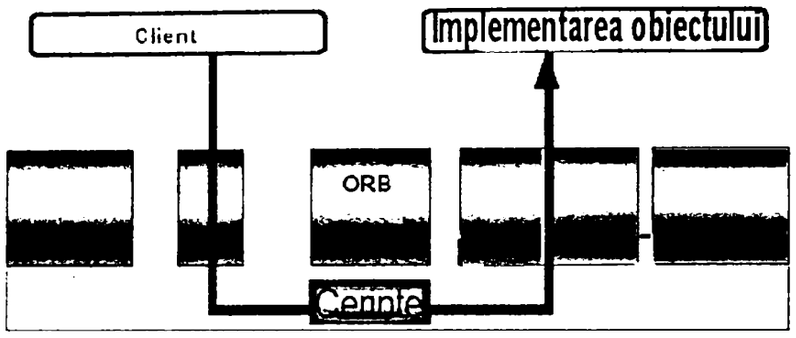


Fig 2.47 Arhitectura ORB

ORB a fost definit de OMG ca Common Object Request Broker Architecture (CORBA). Structura acestuia este prezentată în figura 2.48. CORBA defineşte intefața între obiecte în această arhitectură printr-un limaj specific numit Interface Definition Language (IDL). Sintaxa acestui limbaj este asemănătoare cu sintaxa limbajului C++.

Serviciile obiectului definesc interfețele şi semanticile care sunt folosite în comun în starea obiect distribuit. Ele sunt folosite în comun de către mai mulți utilizatori. Aceştia sunt de fapt obiecte aplicație, cu facilități comune. Ei asigură suport pentru diverse aplicații în domeniul CAD, management de rețea etc.

O modalitate relevantă în care poate fi folosit OMA este modul în care, o aplicație interacționează cu un ODBMS. OMA şi CORBA pot fi folosite pentru a genera conceptul de legătură între aplicații şi DBMS-ul obiectual(figura 2.49).

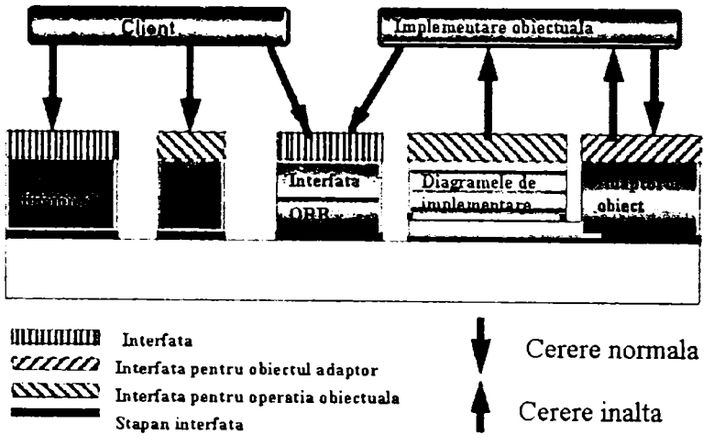


Fig 2.48 Structura CORBA

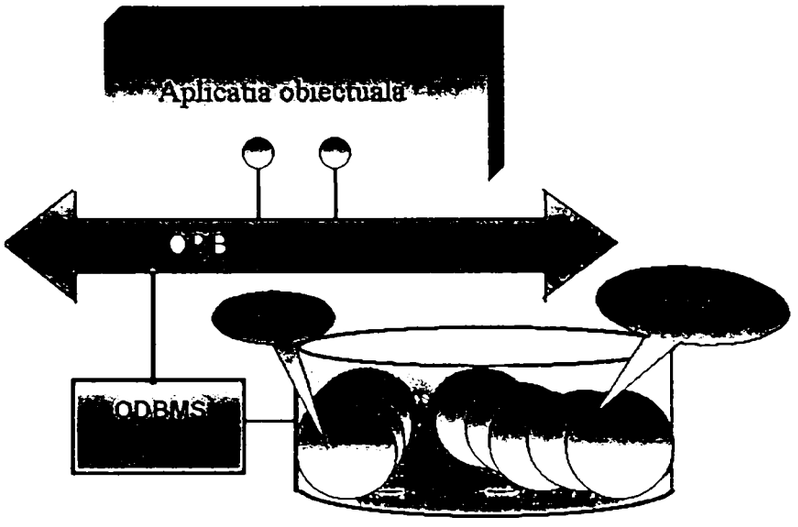


Fig 2.49 Legătura între aplicații folosind ORB şi OMA

O altă cale prin care OMA este relevant, în contextul ODBMS este acela că o colecție de obiecte într-o arhitectură obiect distribuită poate fi considerată ca o bază de date obiectuală ca în figura 2.50

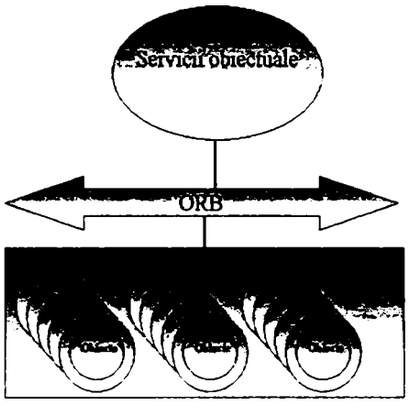


Fig 2.50 ODBMS în standardul OMG

Standardul OMA, CORBA şi COSS este foarte bun atât pentru implementarea ODBMS, cât şi pentru dezvoltarea arhitecturilor obiect distribuite, în care ODBMS-urile pot fi folosite. Datorită acestui fapt, conceptele OMG au fost şi sunt folosite de către mulți producători de software pentru managementul OODBMS.

3

Proiectarea bazelor de date obiectuale

Proiectarea bazelor de date este un proces complex, care trebuie să țină seama de o serie de factori. În lumea dură a software-ului contemporan realizarea unei aplicații este influențată determinant de modul în care este realizată proiectarea sa. Aspectele de care trebuie să se țină cont sunt multiple și sunt influențate de alte domenii cum ar fi: ingineria software, business etc.

Conform lui [Rob 95], proiectarea bazei de date, indiferent de modelul ales, conține în general două aspecte importante și anume:

- Proiectarea logică a bazei de date;
- Proiectarea fizică a bazei de date.

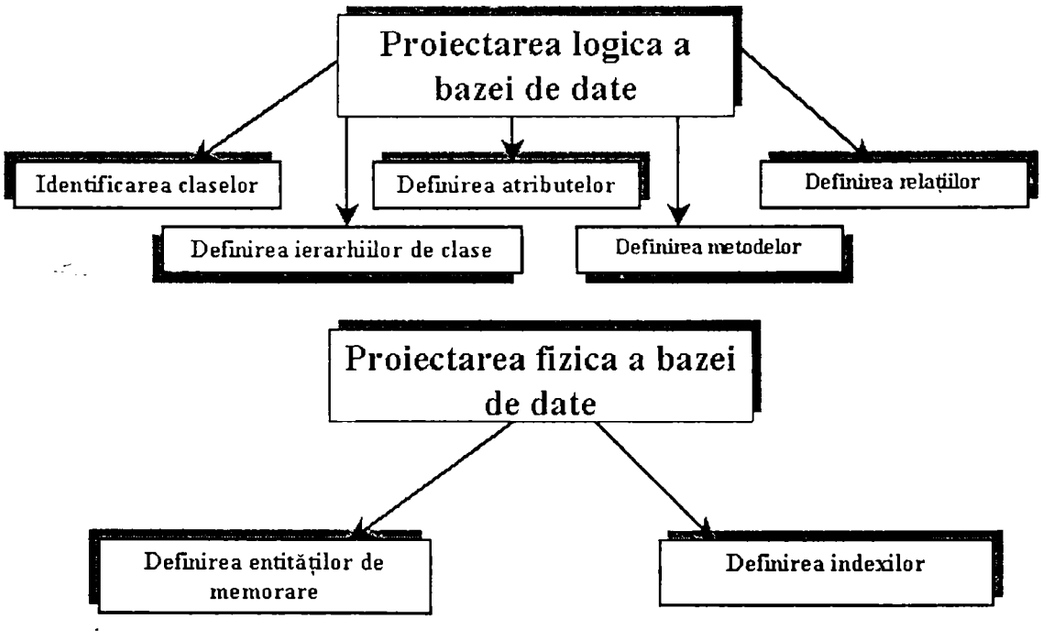


Fig 3.1 Aspectele proiectării bazei de date

În cazul bazelor de date orientate obiect, etapele care trebuie parcurse în activitatea de proiectare, sunt prezentate în figura 3.1.

3.1 Identificarea claselor

Un prim pas în identificarea claselor constă în determinarea obiectelor care vor aparține la o anumită clasă. În acest proces se determină numărul de obiecte ale clasei. Pentru realizarea acestui lucru sunt necesare două procese primare:

- Abstractizarea;
- Descompunerea.

Abstractizarea este procesul în care se identifică dacă două clase au ceva în comun. De exemplu "Alezoare de mână" şi "Cuțite pentru retezat" sunt amândouă cazuri speciale de "Scule Aşchietoare" (figura 2)

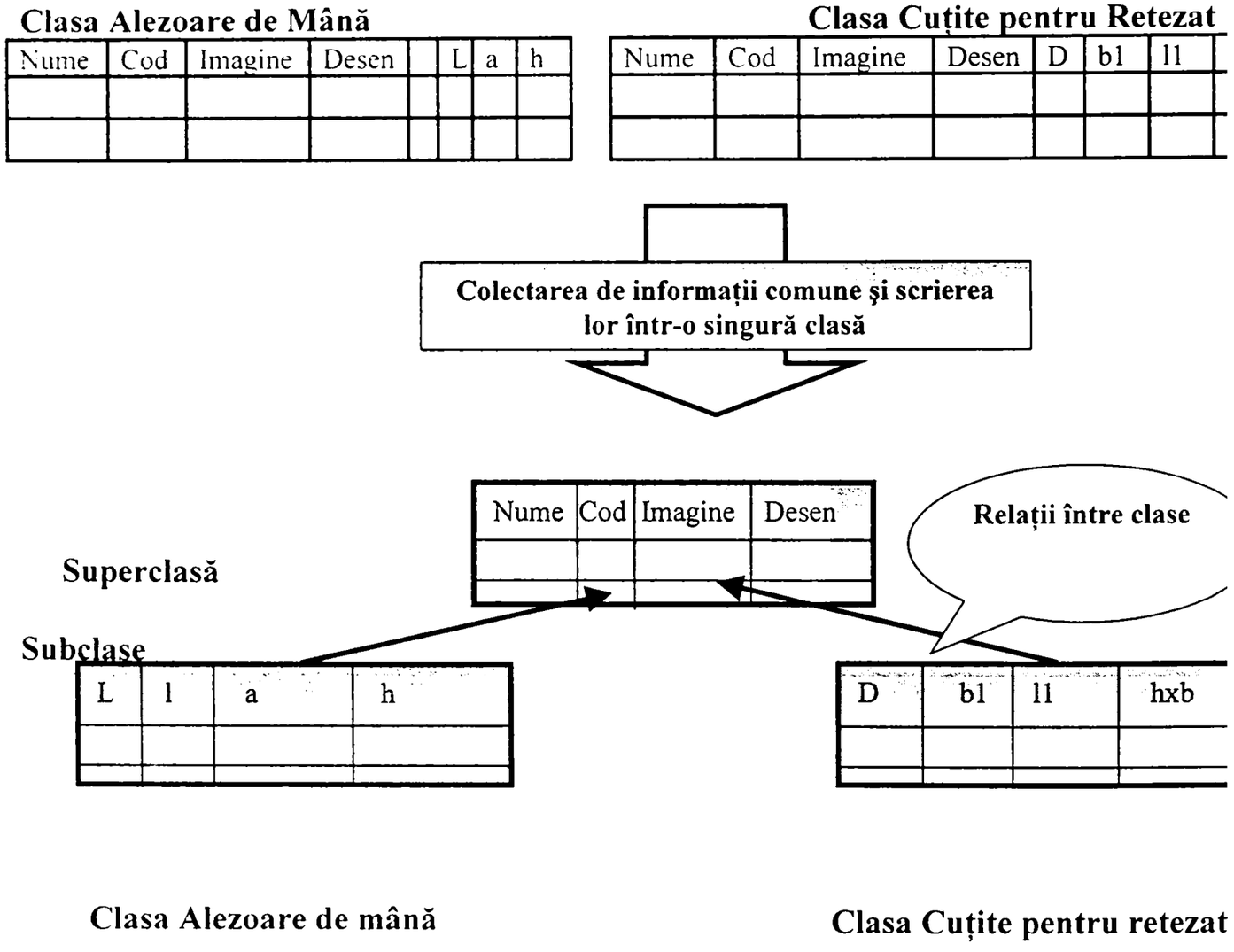


Fig 3.2 Identificarea claselor

"Alezoare de mână" şi "Cuțite pentru retezat" devin subclase ale clasei "Scule aşchietoare", care este definită ca o superclasă pentru amândouă. Informațiile comune pentru amândouă, ca Nume, Cod, Imagine şi Desen pot fi combinate şi definite în superclasa "Scule aşchitoare". "Alezoare de mână" şi

"Cuțite pentru retezat" conțin și informații specifice fiecăreia dintre ele, care nu apar în superclasă. De exemplu "Alezoare de Mână conține informația "L" sau "h", iar clasa "Cuțite pentru Retezat", conține informația "hxb" sau "l1" sau "b1".

3.1.2 Descompunerea

Descompunerea este procesul prin care un obiect se împarte într-un obiect multiplu, pentru reprezentarea unor structuri interne complexe. De exemplu proiectul sculei include un plan de operații, un calcul de pret, o documentare, figura 3.3.

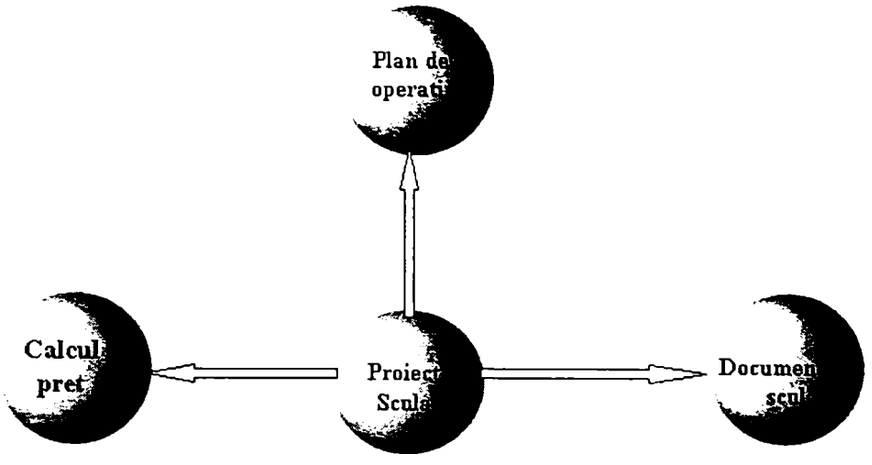


Fig 3.3 Descompunerea obiectelor

Rezultatul descompunerii este identificarea claselor adiționale conectate la clasa originală prin relații. De cele mai multe ori aceasta poate fi o relație de tipul one to many. Procesul de normalizare descris în procesul de analiză al bazei de date, decide care proprietăți aparțin unei anumite clase.

În stagiul de identificare al claselor pot apare câteva probleme tipice ca de exemplu:

- Cum poate fi introdusă informația în obiect?;
- Cum poate fi folosită normalizarea pentru un obiect bază de date?;

În luarea unei decizii privind împărțirea sau combinarea obiectelor, factorii care trebuie luați în considerare sunt:

- Tipul relației: permanentă sau temporară. Dacă relația este temporară atunci este necesar să separam obiectele;
- Relațiile sunt one to one sau sunt one to many? Dacă relațiile sunt one to many cea mai bună soluție este de a separa obiectele.

3.1.3 Normalizarea

În general un soft pentru o bază de date obiectuală permite colecții de valori denumite atribute. De exemplu limbajele de programare cunoscute de către un programator, pot fi introduse într-un atribut şir de caractere.

Pornind de la exemplul prezentat anterior, folosind atribute, o soluție mai bună este de a defini o clasă separată numită "Caracterisici geometrice" şi de a defini, o relație între clasă "Cuțite pentru Retezat" şi clasa "Caracterisici geometrice". Aducerea unor noi atribute la clasa Cuțite pentru Retezat, este apoi un proces mai simplu (figura 3.4).

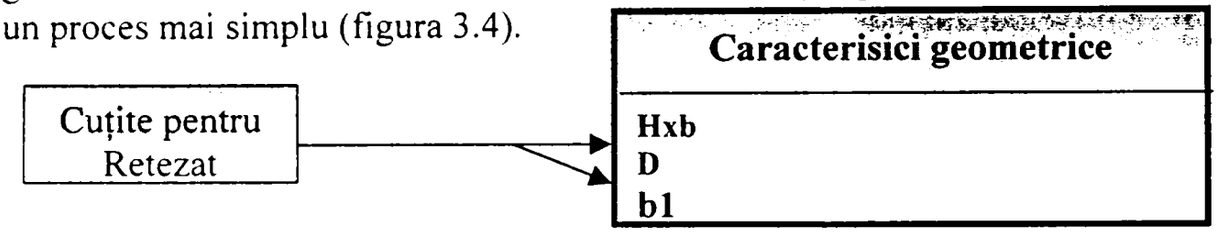


Fig 3.4 Normalizarea

Dacă informația despre limbajul cunoscut de către un programator, este întodeauna obținută printr-o metodă, această poate fi rescrisă în cazul în care limbajul se schimbă fără a afecta interfața. Cu alte cuvinte procesul de normalizare constă, în definirea unor clase corespunzătoare, unei anumite entități şi adăugarea ulterioară de atribute şi metode.

3.2 Definirea ierarhiilor de clase

Într-o aplicație, este necesar să aranjăm clasele într-o ierarhie de clase. Atunci când este definită o ierarhie de clase sunt necesare, a fi luate în considerare următoarele aspecte:

- Principiul incluziunii;
- Avantajele polimorfismului;
- Evitarea migrației instanțelor;
- Folosirea moştenirii multiple;
- Evitarea ambiguității moştenirii multiple;
- Evitarea limitelor sistemului;
- Reutilizarea claselor existente

3.2.1 Principiul incluziunii

Există două căi de a cerceta moştenirea. Una este de a o reflecta într-o relație de apartenență, de exemplu "Cuțit pentru Retezat" este o sculă aşchietoare. Astfel "Cuțit pentru Retezat" moşteneşte caracteristicile clasei "Scule Aşchietoare". Aceasta este denumită *moştenirea inclusă*. O a doua cale este aceea de a scoate în evidență *relația reutilizabilă*. De exemplu considerăm

clasa "Scule aşchietoare" care conține mai multe metode. Definim o clasă nouă numită "Dispozitive". O definire ierarhică, bazată pe conceptul de moştenire nu este corectă, pentru că nu orice dispozitiv este şi sculă aşchietoare.

3.2.2 Avantajele polimorfismului

În activitatea de proiectare a bazelor de date obiectuale, polimorfismul (cap 2 paragraf 2.3), este unul dintre factorii importanți. Conform definiției, dacă în două clase diferite există o metodă cu acelaşi nume, putem considera metoda polimorfică. Polimorfismul este unul dintre cele mai puternice concepte ale tehnologiei obiectuale, pentru că el asigură posibilități de schimbare. În locul scrierii de cod în toate cazurile posibile ale unui algoritm, putem scrie cod separat pentru fiecare clasă de obiecte. Când sunt introduse clase noi în sistem, nu trebuie modificat codul existent ci doar trebuie adăgate noi metode pentru clasele noi. Folosind polimorfismul pot fi identificate subclase ale unor clase existente, cu toate că ele au aceleaşi caracteristici, pot avea metode diferite care să le deosebească ierarhic. Clasa "Caută alezoare" este identică cu clasa "Caută Scule aşchietoare" dar metoda de căutare are introduse subrutine de validare specifice care ne fac să considerăm clasa "Caută alezoare" ca şi subclasă a clasei "Caută Scule aşchietoare". O deosebită importanță are polimorfismul în determinarea ierarhiei de clase. De multe ori acest lucru poate anticipa modificări ale claselor. Acest lucru poate implica introducerea unui nivel nou în ierarhia de clase care nu are nici o importanță pe moment dar poate fi folosit ulterior. De exemplu se poate constata că unele dintre metodele scrise pentru clasa "Scule aşchietoare" pot fi aplicate numai acelor scule care sunt fabricate în societate sau în curs de fabricare. Din această cauză o soluție ar fi împărțirea clasei "Scule aşchietoare" în două clase "Scule aşchietoare fabricate" şi "Scule aşchietoare în curs de fabricare" cu metodele specifice fiecărei clase. Aceasta duce la extinderea ierarhiei de clase şi la folosirea ulterioară a polimorfismului metodelor.

3.2.3 Evitarea necesității de migrare a instanțelor

După definirea ierarhiilor de clase poate apare situația în care un anumit obiect este necesar să-şi schimbe clasa. În anumite condiții acest lucru poate avea însă efecte nefericite asupra aplicației, astfel pe cât posibil se va căuta evitarea acestui lucru.

3.2.4 Folosirea moştenirii multiple

Unii specialişti [Booch 94] nu recomandă folosirea moştenirii multiple (cap. 2 paragraf. 2.1.8). Moştenirea multiplă apare în mod natural când există

două definiții ortogonale pentru același obiect. De exemplu putem clasifica oamenii de gen feminin și masculin sau ca adulți și copii. Putem construi ierarhia de clase (figura 3.5).

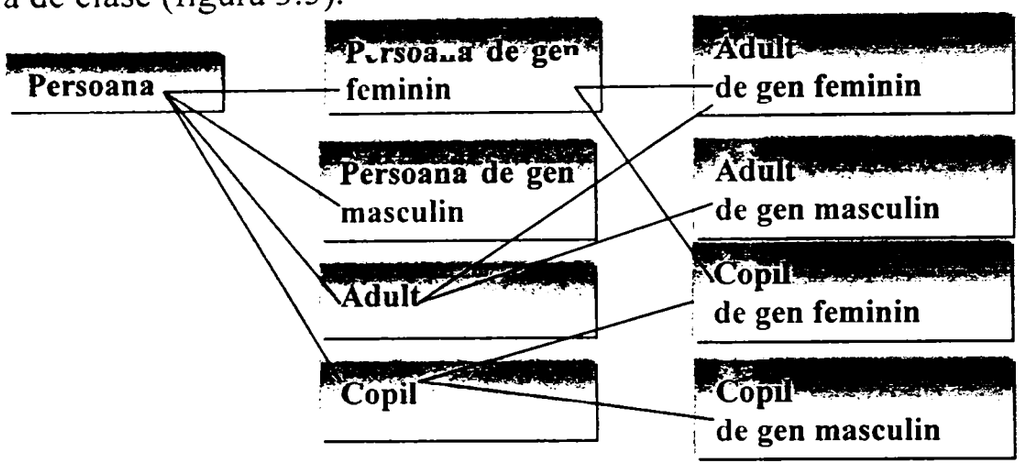


Fig 3.5 Ambiguitatea moștenirii multiple

Ierarhia de clase este prezentată pe două nivele. Toate instanțele ierarhiei, pot să aparțină la una dintre cele patru clase de pe nivelul de jos, de exemplu copii de gen masculin. Pericolul acestui concept este acela că el poate deveni foarte dificil. Numărul de clase poate crește foarte repede și este dificil de testat toate posibilitățile de interacțiune ale metodelor polimorfice. Înainte de a folosi moștenirea multiplă este necesar de a lua în considerare următoarele aspecte:

- Rafinarea claselor până la cel mai înalt grad se va face numai dacă acest lucru este absolut necesar. Pentru exemplul nostru se va analiza cazul în care este necesară sau nu existența clasei adulți și a clasei copii sau numai a clasei "Persoane de gen barbătesc";
- Trebuie verificat dacă legăturile dintre clase respectă principiul incluziunii. Este foarte tentant de a folosii moștenirea multiplă pentru a reprezenta în totalitate o anumită relație;
- Clasificarea trebuie să fie stabilă, pentru că tendința obiectelor de a-și schimba clasa poate influența negativ o anumită aplicație;

Cazurile în care poate fi folosită moștenirea multiplă sunt acelea în care pot fi definite clasele ce pot acoperi diverse categorii distincte între ele. Acest concept poate fi folosit în activitatea de proiectare a unor medii pur obiectuale, de exemplu Jasmine dar poate crea ambiguități care trebuie evitate tocmai în această activitate. Datorită acestui fapt, alte medii de programare, de exemplu Visual Fox nu are implementat acest concept.

3.2.5 Evitarea ambiguităților în moștenirea multiplă

În Jasmine de exemplu, unul dintre cazurile defavorabile care poate apare la folosirea moștenirii multiple, este acela în care superclasele au același nume

de proprietăți, sau de metode. Nu toate mediile orientate obiect suportă moştenirea multiplă. Din acest punct de vedere, de multe ori ea nu este luată în considerare. În cazul în care mediul de programare permite moştenirea multiplă, în primul rând trebuie identificată în mod corect prin rafinare, fiecare metodă sau proprietate. În general această problemă poate fi evitată prin folosirea unor nume diferite pentru proprietăți sau metode cu acelaşi înțeles.

3.2.6 Evitarea limitelor sistemului

Limitele unui anumit sistem se pot referi la numărul de clase care este permis, precum şi al numărului de subclase ale fiecărei clase. Mediul Jasmine, [Fallo 99] are posibilitatea de a defini o entitate (Store), care permite, specificarea zonei în care va fi realizată baza de date. Această zonă are o anumită mărime în MO şi un număr de pagini. Funcție de aceste mărimi sunt definite limitele sistemului.

3.2.7 Gruparea claselor în familii de clase

În unele medii de programare bazele de date orientate obiect (Jasmine), [Fallo 99-2], este permisă definirea unei familii de clase, de exemplu "Sistemul de fabricație al sculelor", unde vor fi definite clasele aplicației utilizator. Pentru definirea familiilor de clase există două căi:

- Se poate face o definire de clasă în primul stadiu de proiectare, în momentul în care este definită modularitatea sistemului proiectat. De exemplu o aplicație este împărțită ca sistem în două părți: una care se ocupă publicarea documentelor şi o altă parte care asigură facturarea unor produse. Din prima fază a proiectării pot fi alocate familii de clase, fiecărui termen în mod separat, fără a se face definirea în amănunt a fiecărei clase;
- Se poate defini ierarhia de clase, în amănunt şi apoi se trece la împărțirea claselor în familii de clase, pe baza unor caracteristici comune.

3.2.8 Refolosirea claselor existente

Un important beneficiu al tehnicii obiectuale este refolosirea claselor care există deja. Existența claselor şi refolosirea acestora are următoarele avantaje:

- Definirea unei subclase, a unei clase deja existente. Cu ajutorul acestui concept putem să definim o clasă care împrumută toate caracteristicile superclasei şi căreia îi putem definii ulterior alte proprietăți;
- Definirea relațiilor între clasele existente;

- Copierea şi modificarea codului sursă.

3.3 Definirea atributelor şi metodelor

În general operațiile de actualizare şi ştergere a informațiilor, se pot face fie cu ajutorul atributelor, fie cu ajutorul metodelor. Este de preferat folosirea unei metode de compromis, prin care utilizatorii pot citi atributele şi pot să le modifice conținutul cu ajutorul metodelor. Aceasta permite scrierea unor metode care să impună reguli de validare pentru valori noi, ceea ce va impiedica apariția de date redundante sau date greşite în baza de date.

3.3.1 Supraîncărcarea atributelor

Supraîncărcarea apare acolo unde un atribut poate fi dedus din alt atribut, de exemplu durabilitatea unei scule poate fi dedusă din cod. Aceata este o problemă serioasă care poate apare în baze de date complexe, pentru că aplicațiile fac presupuneri asupra datelor care pot fi invalidate de către anumite schimbări. Metoda prin care se poate rezolva acest aspect pentru exemplul prezentat, este de a defini o metodă în clasa "Scule aşchietoare" prin care să fie returnată durabilitatea unei anumite scule.

3.3.2 Atribute complexe

Una dintre principalele restricții ale bazelor de date relaționale este faptul că atributele complexe sunt interzise. Dacă în teoria relațională un atribut complex, ca de exemplu adresa poate fi "spart" în mai multe atribute, în bazele de date obiectuale un atribut complex, poate fi prelucrat folosind metode care să extragă din el doar partea care prezintă interes.

3.3.3 Gestionarea tipurilor abstracte de date

În multe cazuri este necesară reprezentarea atributelor folosind tipuri abstracte de date, la care reprezentarea internă este invizibilă. De exemplu tipuri de date abstracte care pot fi întâlnite adesea sunt:

- Măsuri (longitudine sau latitudine);
- Tipuri de date multimedia ca de exemplu sunete şi imagini.

În general există două căi pentru reprezentarea lor:

1. definirea unei clase;
2. căutând o reprezentare literală.

Prin prima metodă datele abstracte se pot reprezenta cu ajutorul definițiilor de clase, iar prin cea de a doua cale anumite valori pot fi ascunse folosind metode adecvate în acest scop.

3.4 Definirea relațiilor

Factorii care trebuie luați în considerare [Rob 95] atunci când sunt necesare relații între obiecte sunt următorii:

- Menținerea avantajului pe care îl permite cheia primară folosită în bazele de date relaționale;
- Folosirea unei căi alternative pentru menținerea relațiilor de tip one to many și many to many;
- Menținerea integrității referențiale.

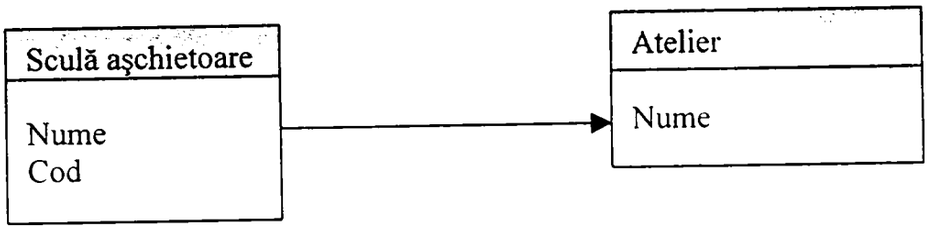


Fig 3.6 Relații între clase în procesul de proiectare

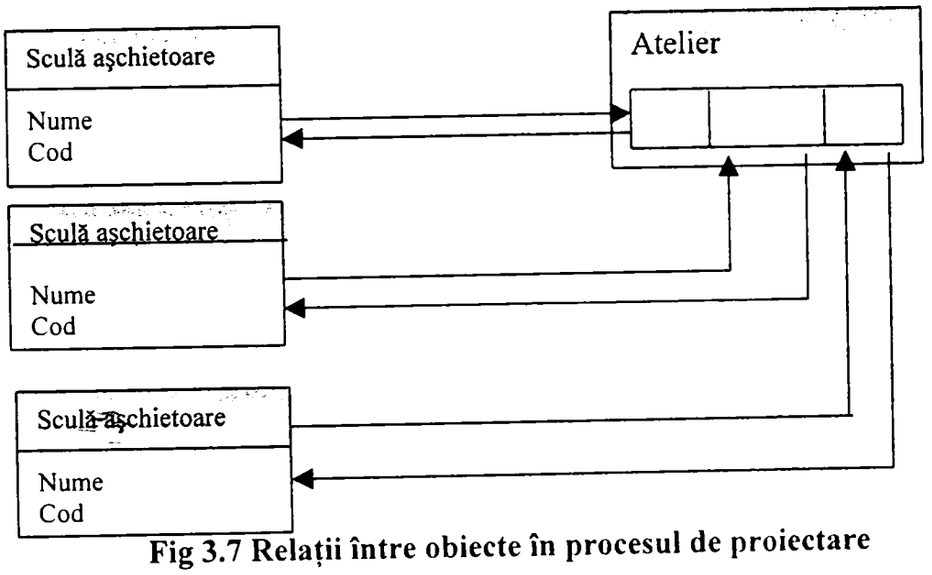


Fig 3.7 Relații între obiecte în procesul de proiectare

3.4.1 Relații de tip one to many

Acest tip de relații au fost definite în cap. 2 paragraf 2.4.3 şi se stabileşte între două sau mai multe obiecte. Pentru aceasta se folosesc idetificatori interni ai obiectului care fac referire la unul sau mai multe obiecte. Astfel pentru a reprezenta o relație de tip one to many este necesar ca într-o clasă să fie definit un identificator care localizează mai multe obiecte dintr-o altă clasă. De

exemplu o interogare în clasa Scule aşchietoare poate să caute toate sculele dintr-un atelier. Aceasta este metoda cea mai simplă de reprezentare (fig. 3.6)

A doua opțiune este de a menține referința în ambele direcții (figura 3.7) şi menținerea referinței de tipul one to many (figura 3.9).

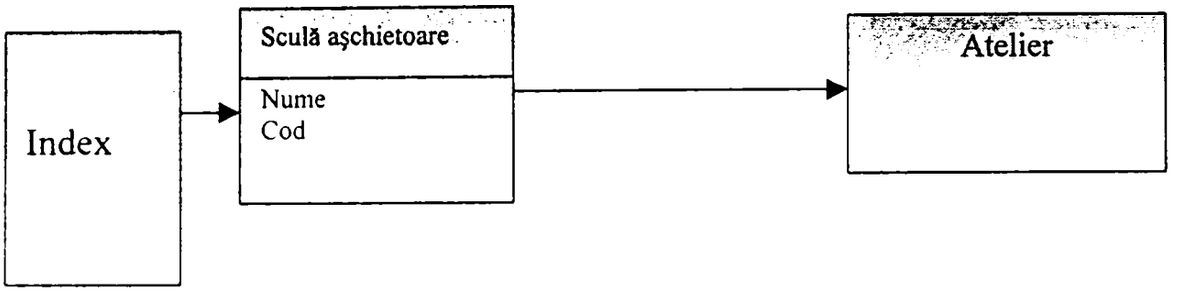


Fig 3.8 Index pentru menținerea unei relații

O altă metodă este aceea prin care se poate folosi un index pentru menținerea unei relații de tip one to many (figura 8)

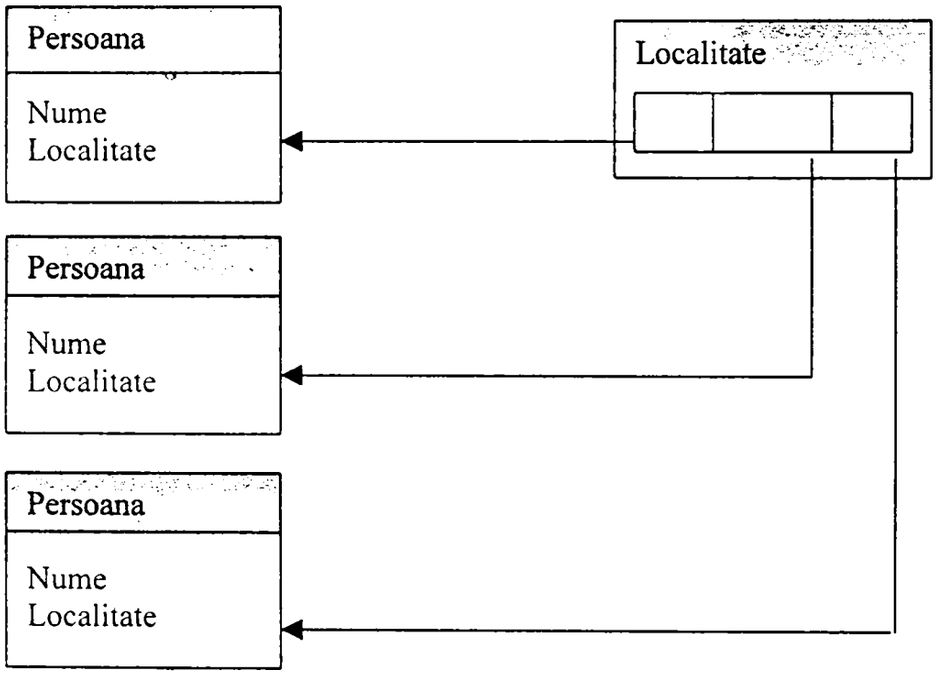


Fig 3.9 Menținerea referinței de tip unul la mai multe

Datorită faptului că în acest caz căutarea infomațiilor este mult mai rapidă este de recomandat folosirea acestei metode de menținere a relației de tip one to many. Ca şi în cazul atributelor relațiile dintre obiecte pot fi ascunse folosind metode.

3.4.2 Relații de tip many to many

Acest tip de relație poate fi reprezentat în OODBMS-ul Jasmine direct folosind colecții de valori implementate sub forma unor proprietăți. Înainte de a reprezenta o asıfel de relație trebuie avut în vedere că nici o dată nu este asociată

cu această relație. În general o astfel de relație se poate implementa folosind două sau mai multe relații de tip one to many. La fel ca și în cazul celorlate tipuri de relații pot fi folosite metode, pentru a ascunde implementarea unei astfel de relații.

3.5 Integritatea referențială

Termenul *integritate referențială înseamnă din punct de vedere obiectual, că un obiect nu poate referi niciodată un obiect care nu există.* Aceasta înseamnă că în cazul în care obiectul este șters, toate referințele la el trebuie șterse. Integritarea referențială poate fi implementată folosind metode [Blaha 95]. Definirea metodelor ține cont de următorii factori:

- Polimorfismul;
- Folosirea metodelor pentru accesarea unor sisteme externe bazei de date;
- Definirea clasei la care trebuie să aparțină o anumită metodă;
- Transformarea unei anumite funcții sau proceduri într-o anumită metodă.

Cele mai uzuale tipuri de metode folosite de către utilizatori sunt:

- Pentru obținerea de informații din baza de date;
- Actualizare informațiilor din baza de date obiectuală;
- Obținerea de rapoarte referitoare la diverse activități din baza de date.

Există diverse tehnici care sunt folosite pentru manipularea integrității referențiale:

- să nu se permită utilizatorului să șteargă explicit obiectele, adică sistemul șterge automat obiectele care nu mai sunt accesibile utilizatorului;
- să se permită utilizatorului să șteargă obiectele când acestea nu mai sunt cerute. În acest caz sistemul poate detecta automat referințele care nu sunt valabile și le atribuie valoarea NULL sau nu permite ștergerea;
- să se permită utilizatorului să modifice și să șteargă obiectele și relațiile atunci când ele nu mai sunt necesare

3.6 Folosirea metodelor pentru accesarea sistemelor externe

Unele metode scrise într-un anumit limbaj de programare pot accesa diverse medii externe ca de exemplu:

- Fișiere sistem, poșta electronică, serviciile Internet;
- Cereri SQL de accesare a unor baze de date relaționale, atât local cât și la distanță

Aceste metode trebuie să țină cont de modul de desfăşurare al tranzacțiilor şi mediul soft pentru o astfel de bază de date trebuie să aibă implementată tehnologia de tratare a tranzacțiilor.

Referitor la metode este important din punct de vedere al proiectantului, să permită folosirea polimorfismului. Aceasta înseamnă că putem avea o singură operație cu un anumit nume, care poate fi implementată în diferite feluri funcție de clasa de obiecte. Cu alte cuvinte în diferite clase putem avea metode cu acelaşi nume. Folosirea polimorfismului elimină folosirea codului inutil.

3.7 Proiectarea fizică a bazei de date

Conform lui [Rob 95], proiectarea fizică a bazei de date constă în:

1. Definirea suportului de memorare
2. Definirea indexilor

3.7.1 Definirea suportului de memorare

În general proiectarea bazei de date într-un mediu obiectual, urmăreşte mai mult sau mai puțin regulile de proiectare expuse anterior. Proiectarea în mediul obiectual este însă total diferită fată de aceea într-un mediu relațional sau într-un mediu hibrid.

Proiectarea într-un mediu hibrid poate considera obiectele de date relațiile bazei de date urmând a se stabilii ulterior modul de reprezentare al acestora în cadrul claselor.

Proiectarea bazei de date într-un mediu pur obiectual este total diferită. Dispare noțiunea de relație, obiectele putând memora datele de o anumită natură prin proprietățile pe care le au. Manipularea obiectelor se realizează prin metode scrise într-un limbaj specific sau prin interogări adaptate mediilor obiectuale.

Proiectarea fizică a bazei de date obiectuale, depinde foarte mult de mediul în care se lucrează. Într-un mediu pur obiectual soluția aleasă poate consta în folosirea unui singur fişier pentru alegerea bazei de date sau a mai multor fişiere dacă proiectantul consideră că în acest mod se asigură un management mai bun pentru baza de date proiectată.

3.7.2 Definirea indexilor

Indexii sunt asemănători cu cei din bazele de date relaționale. În general ei sunt definiți prin proceduri proprii mediului specific pentru baza de date obiectuală. Ei sunt destinați pentru a optimiza interogările

3.8 Cerințele unui model de date pentru o bază de date obiectuală a unui sistem de fabricație

1. Datele trebuie să fie modelate ca obiecte, fiind organizate în ierarhii şi trebuie să respecte agregarea şi generalizarea;
2. Modelul de date trebuie să permită definirea proprietăților obiectelor;
3. Modelul de dată trebuie să permită definirea metodelor unui obiect;
4. Modelul de date trebuie să suporte moştenirea proprietăților şi a metodelor;
5. Modelul de date trebuie să permită stabilirea relațiilor dintre clasele de obiecte şi între obiecte;
6. Trebuie să permită modificarea dinamică a obiectelor dacă acest lucru este necesar în procesul de proiectare al bazei de date;
7. Trebuie să permită suport pentru structuri recursive de obiecte;
8. Să permită posibilități eficiente şi flexibile pentru actualizarea obiectelor;
9. Opțional poate permite moştenirea multiplă;
10. Suport pentru proceduri şi metode. Metodele operează într-un sistem mecanic asupra unei mulțimi de obiecte şi au ca efect producerea unei mulțimi de obiecte. În astfel de cazuri de multe ori nu este foarte clar care metode acționează asupra unor obiecte. De aceea este necesar definirea conceptului de procedură, distinct față de acela de clasă;
11. Păstrarea constrângerilor de integritate ale datelor.

3.9 Proiectarea deciziilor pentru un OODBMS al unui sistem de fabricație

Deciziile care se iau în procesul de proiectare trebuie să țină seama de cerințele sistemului de fabricație pentru care se face aceasta.

- Prima decizie constă în a respecta paradigma de modelare a datelor importante şi aceasta poate consta în faptul că, clasele trebuie să fie definite riguros şi fiecare obiect trebuie să fie o instanță a unei anumite clase.
- A doua decizie importantă constă în faptul că limbajul de programare ales trebuie să fie orientat obiect.
- Restul deciziilor în aceată fază a proiectării bazei de date se referă în special la asigurarea moştenirii multiple, dacă este cazul, la asigurarea structurii recursive a obiectelor, la folosirea unor clase abstracte care să asigure descrierea proprietăților şi operațiilor subclaselor.

În general, la nivel de societate economică, proiectarea orientată obiect trebuie să conțină o proiectare comportamentală, care implică o analiză detaliată a cerințelor de prelucrare a societății comerciale. În analiza orientată pe obiecte, cerințele de prelucrare sunt transpuse într-un set de metode, care sunt unice pentru fiecare clasă. *Metodele publice* adică acelea care sunt vizibile trebuie să

fie diferite de *metodele private* care sunt metode interne proprii fiecărei clase. Există trei tipuri de metode publice şi private:

- Constructorii şi destructorii care generează noi instanţe a unei clase, respectiv şterg instanţele ce nu mai sunt necesare;
- Metodele de acces care returnează valoarea unui atribut sau set de atribute al unei instanţe a unei clase;
- Metode de transformare care schimbă starea unei instanţe a unei clase.

4

Proiectarea schemelor obiectuale (externe) şi a claselor derivate

În cazul bazelor de date relaționale schema entitate relație este modul concret prin care este finalizată activitatea de proiectare a bazei de date. Tot astfel activitatea de proiectare a unei baze de date obiectuale are ca punct final al activității de proiectare realizarea schemelor externe corespunzătoare OODBMS.

Funcțiile unei baze de date pentru pentru un DBMS se clasifică în general pe trei nivele: logic, fizic, extern.

Pentru fiecare nivel informația este reprezentată de către schema conceptuală, internă şi externă a bazei de date. Schemele obiectuale oferă o imagine asupra informației conținută în *schema conceptuală*. Aceasta permite utilizatorilor să prezinte într-o reprezentare logică datele, conform cerințelor lor. Schemele obiectuale prevăd independența logică a informației (multe aspecte ale schemei conceptuale pot să fie schimbate fără a modifica vederile schemei conceptuale, oferite de schemele obiectuale). Scopul principal al schemelor externe este acela de a furniza un mecanism care simulează schimbările care pot avea loc în schema conceptuală. Informațiile din schemele obiectuale trebuie să provină din schema conceptuală.

Arhitectura pe trei nivele denumită şi ANSI/SPARC se adresează în mare măsură bazelor de date relaționale. O cerință importantă a unui OODBMS este independența logică a obiectelor

În OODBMS, schemele obiectuale pot să conțină clasele de la schema conceptuală, precum şi *clase derivate,* definite din clasele existente. Aceste clase oferă informații despre datele conținute în clasele din care ele au provenit. Definiția claselor derivate, este un concept important în definiția schemelor externe în OODBMS.

O altă direcție importantă de cercetare, este definirea mecanismului de reprezentare a schimbărilor, care se pot efectua în schemele obiectuale.

4.1 Scheme externe şi clase derivate

4.1.1 Definirea schemei obiectuale şi a schemei interne

Teoria ANSI/SPARC propune o arhitectură cu trei nivele pentru DBMS prezentată în figura 4.1. Schema conceptuală este reprezentarea modelului real al bazei de date. *Fiecare schemă externă este derivată din schema conceptuală şi descrie anumite informaţii care se adresează unui grup de utilizatori.* Bazele de date care au scheme externe sunt flexibile şi adaptabile la schimbări funcţie de modul în care utilizatorii interpretează datele. Schemele obiectuale prevăd independenţa logică a datelor. Multe aspecte ale schemei conceptuale pot să fie schimbate fără a trebui să fie modificate vederile schemei conceptuale, oferite de schemele obiectuale.

Schema internă este reprezentarea fizică a datelor memorate în baza de date. Ea specifică care date sunt memorate în baza de date şi modul cum sunt acestea memorate. Deosebirea dintre schema conceptuală şi schema internă constă în independenţa fizică a datelor, adică multe aspecte ale implementării fizice pot fi modificate, fără a fi afectată viziunea abstractă asupra bazei de date.

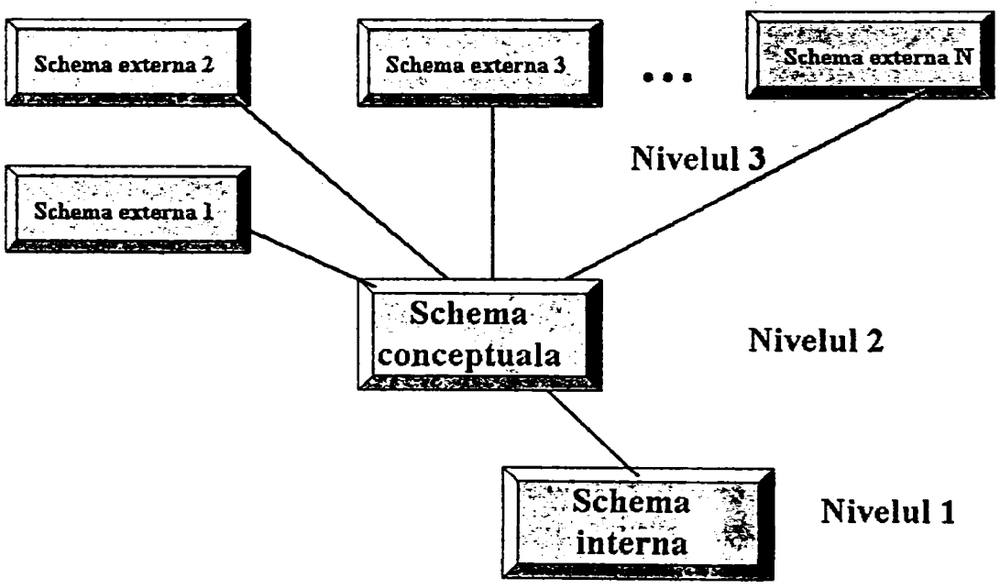


Fig 4.1 Arhitectura ANSI/SPARC pe trei nivele

4.1.2 Legăturile dintre schema conceptuală şi schemele externe

În OODBMS schema conceptuală şi schema externă sunt scheme obiect, definite conform paradigmei obiectuale. În acest sens, din punct de vedere al utilizatorului nu sunt deosebiri privind modul de lucru cu schema obiectuală sau

cu schema externă, datele fiind organizate la fel în ambele scheme. Informația conținută în schema conceptuală este reprezentată prin clase. Schemele externe, sunt derivate din schema conceptuală. Informația conținută în fiecare schemă externă este derivată din informația definită în schema conceptuală, dar nu este obligatoriu ca, clasele definite în schema externă să fie definite anterior în schema conceptuală. În schema externă pot fi definite clase noi numite clase derivate, definite direct sau indirect din clasele de bază, adică cele existente în schema conceptuală. Aceste clase sunt definite şi incluse în dicționarul de date. Clasa derivată poate reprezenta un concept relevant în schema externă dar acest concept nu trebuie scos în evidență în schema conceptuală. În metodologia care va fi prezentată în continuare vor fi luate în considerare două aspecte importante şi anume:

- schema conceptuală şi schema externă sunt scheme obiect definite conform paradgimei orientat obiect;
- schemele externe pot conține atât clase ce aparțin schemei conceptuale cât şi clase derivate din schema conceptuală şi care nu aparțin acesteia.

4.1.3 Organizarea dicționarului de date şi definirea schemei sistem externe

Definirea în OODBMS, a schemei conceptuale şi a schemei externe, se bazează pe dicționarul de date, prezentat în figura 4.2.

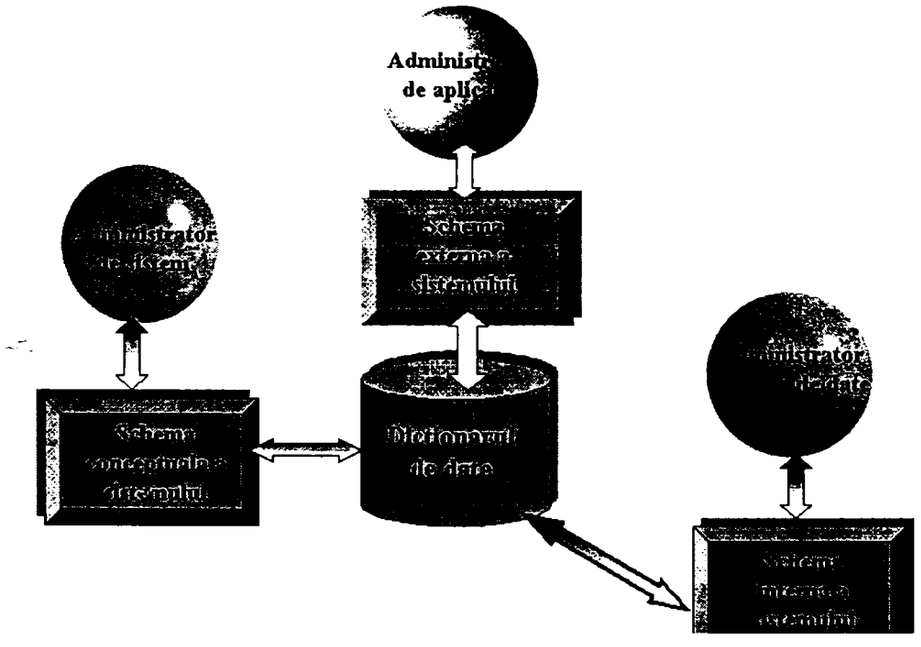


Fig 4.2 Dicționarul de date

Informația conținută în schemele externe, trebuie să provină din informația conținută în schema conceptuală.

Schemele obiectuale pot avea aceeaşi structură ca şi schema conceptuală a OODBMS. *Din această cauză putem considera schema externă ca fiind identică cu schema conceptuală sau schema obiectuală.* În general terminologia ANSI/SPARC nu se foloseşte în cazul OODBS.

Dicţionarul de date conţine toate informaţiile referitoare la managementul şi utilizarea unui sistem bază de date. Informaţia face referire la schema conceptuală, schema internă şi schemele externe ale bazei de date.

Într-o bază de date există două tipuri de utilizatori ai informaţiei şi anume administratorul de sistem şi administratorul de aplicaţie. Administratorul de sistem defineşte schema conceptuală a sistemului, pe când administratorul de aplicaţie foloseşte schema externă. De aceea informaţia din dicţionarul de date trebuie să fie organizată în scopul de a îndeplini cerinţele acestor utilizatori.

Clasele derivate reprezintă o parte din informaţia conţinută în schema conceptuală sub forma unor clase noi. Ele sunt definite din clasele existente folosind interogări. Dacă în schema externă sunt definite clase noi, acestea sunt integrate în dicţionarul de date împreună cu definiţia schemei externe. Modul de integrare al claselor în schemă poate fi de două feluri:

- folosind moştenirea;
- folosind alte tipuri de relaţii.

O modalitate de a obţine clase derivate din clasele definite în dicţionarul de date este folosirea relaţiilor derivate, definite în [Berti 92], [Monk 94], [Kim 95], [Guerr 97].

Schema conceptuală este conţinută în dicţionarul de date împreună cu definirea claselor derivate şi cu definirile schemelor externe. În dicţionarul de date fiecare clasă derivată este legată printr-o relaţie derivată cu clasele din care a fost definită. Relaţia apare doar în dicţionarul de date.

Pentru a putea defini schemele externe, este necesară definirea schemei sistem externe care se suprapune peste dicţionarul de date. Aşa cum s-a arătat anterior, pentru fiecare parte a bazei de date este necesară definirea propriei scheme externe. Aceasta poate fi definită în două moduri:

1. manual în care fiecare componentă este definită de către utilizator;
2. automat în care componentele schemei externe sunt definite de către sistem fără intervenţia uilizaorului

4.2 Scheme externe (obiectuale)

Schemele obiectuale au aceeaşi structură ca şi schema conceptuală din care ele provin, astfel încât putem să folosim termenul de schemă obiect. Cerinţele pe care trebuie să le îndeplinească schemele obiectuale poartă numele de *proprietatea de închidere [Dayal 89]*, adică fiecare clasă referenţiată de către o clasă inclusă în schemele obiectuală, trebuie să fie introdusă în schema

obiectuală respectivă. *O clasă este referențiată de altă clasă dacă ea apare ca argument a unei metode sau ca domeniu al unui atribut al unei clase secundare.*

4.2.1 Metodologii de definire a schemelor obiectuale

Metodologiile de definire a schemei obiectuale sunt diverse şi ele au apărut o dată cu conceptul de bază de date obiectuală. În ordine cronologică descrescătoare funcție de anul de apariție vor fi prezentate câteva metode.

În [Guerr 97], se propune o metodologie, care permite definirea schemei obiectuale folosind terminologia ANSI pe trei nivele.

Procesul de definire al schemei externe (obiectuale) conține următorii paşi:

- Se foloseşte schema conceptuală sau o altă schemă externă definită anterior şi se definesc clasele derivate necesare. Toate clasele ce vor fi incluse în noua schemă externă trebuie să fie clase derivate noi din aceeaşi schemă. Clasele derivate sunt conectate cu clasele existente prin relații noi denumite *vederi derivate*. Moştenirea relațiilor între clasele derivate este definită în momentul definirii claselor. Corectarea moştenirii între clasele derivate este asigurată de către sistem. Limbajul de definire al claselor derivate permite efectuarea acestor verificări asupra moştenirii.
- Schema obiectuală trebuie să fie închisă. Clasele din schema originală, referite şi neincluse în schema obiectuală, sunt automat redefinite ca noi clase derivate şi incluse în schema obiectuală.

În concepția [Guerr 97] schema conceptuală este denumită *schema de bază.* Clasele derivate se numesc *vederi* şi pentru schemele obiectuală este folosit termenul *schema vederilor.*

Kim & Kelly fac de asemenea referință la arhitectura pe trei nivele, dar terminologia ANSI nu este folosită în totalitate [Kim 95]. Termenul vedere este folosit cu înțelesul de clasă derivată şi poate semnifica totodată schema obiectuală. Procesul de definire al schemei externe este următorul:

- Definirea claselor derivate este necesară pentru definirea schemei obiectuale. Clasele derivate sunt definite din clasele existente. Ele sunt legate cu clasele din care provin prin relații noi denumite *derivat din.* Mulțimea claselor astfel definită formează o ierarhie de moşteniri separate. Ierarhia poartă denumirea de *vederea ierarhiilor derivate.* Clasa derivată este legată prin moştenire cu clasa indicată la momentul definirii sale;
- Selecția claselor care va compune schema obiectuală nu este foarte clar definită. Schema obiectuală poate fi compusă din clase derivate sau nederivate. Clasele derivate şi nederivate au ierarhii de moştenire

separate dar ele pot fi legate prin agregare. Domeniul proprietăților unei clase derivate, poate fi o clasă derivată sau nederivată, dar domeniul proprietăților unei clase nederivate nu poate fi o clasă derivată.

Astfel în schema conceptuală [Kim 95], numită *schema bazei de date* există două structuri separate: una pentru clase şi una pentru vederi sau clase derivate. Cu alte cuvinte clasele derivate sunt integrate în schema conceptuală folosind o nouă relație nedefinită în paradigma orientată obiect.

Rundensteiner, a propus o metodologie de definire a schemei externe cu numele *multivedere (Multiview)* [Runde 92-1]. Metodologia este completată prin algoritmi care au fost prezentați în [Runde 92-3] şi [Runde 92-4]. În această metodologie, specificarea unei scheme externe este împărțită în următoare acțiuni independente:

- Definirea claselor derivate are nevoie de existența unor clase inițiale derivate sau nederivate;
- Integrarea automată a definiției claselor derivate cu toate clasele existente în schema obiect. Acest proces de integrare este realizat folosind moştenirea. Obiectivul este de a menține toate relațiile de moştenire între clasele derivate şi nederivate;
- Selectarea unui set de clase (derivate sau nederivate) pentru care poate fi compusă schema obiect;

Metodologia folosită pentru generarea automată a schemei obiect are două părți şi anume:

- Toate clasele care sunt folosite de către clasele inițiale, sunt incluse în schema obiect. Aceste clase sunt automat adăugate la setul de clase selectat;
- Generarea ierarhiei de clase pentru clasele obținute. Procesul de generare constă în definirea relațiilor de moştenire care există între mulțimea de clase.

În această metodologie clasele derivate se numesc *clase virtuale*, clasele nederivate se numesc *clase de bază*, schemele obiectuale se numesc *vederi*, *scheme vederi* sau *scheme virtuale* şi clasele incluse în schemele obiectuală se numesc *clase vederi*. În acest caz sunt definite două scheme: *schema de bază* care este schema obiect inițială unde toate clasele corespund claselor nederivate şi *schema globală* care este o extensie a schemei de bază şi care cuprinde toate clasele derivate. Schema globală nu este un dicționar de date pentru ca ea conține numai schema de bază la care au fost adăugate noile clase derivate.

O altă metodă este cea definită de către Abiteboul & Bonner. Conform acestor autori [Abite 91], paşii necesari pentru definirea unei scheme externe sunt următorii;

- Selecția claselor, din alte scheme (externe sau conceptuale) pentru a fi incluse în schema dorită. Selecția este făcută folosind

mecanismul de importare. Când clasele sunt importate aceste devin vizibile împreună cu subclasele lor;

- Ascunderea claselor nedorite sau a proprietăților în clasele schemei.

Schema obținută după acești doi pași poate fi modificată ulterior astfel:

- Definirea claselor derivate din clasele incluse în schemele obiectuală. La acest nivel există două mecanisme pentru definirea claselor și anume: generalizarea și specializarea. Clasele derivate sunt automat integrate în schemă, datorită relațiilor de moștenire pe care le au cu clasele de bază.
- Ascunderea proprietăților adiționale sau a claselor în schema obiectuală.

O primă metodă de definire a schemei obiectuale, a fost făcută de către Tanaka, Yoshikawa & Ishiara, în [Tanak 93]. Acest proces fiind denumit virtualizarea schemei. Sistemul lor permite definirea claselor derivate prin semantica obiect conservat, adică ea conține numai obiecte ale clasei de bază. Schema obiectuală conține una sau mai multe clase derivate. Construcția schemei presupune următorii pași:

- Definirea tuturor claselor derivate în schemele obiectuală;
- Definirea manuală a tuturor relațiilor de moștenire între clasele derivate.

După acești pași se obține o schemă externă. Opțional se mai pot desfășura următoarele acțiuni:

- Ștergerea unei subscheme din schema existentă;
- Pentru a obține o nouă schemă se poate importa o schemă existentă anterior și atașarea acesteia, cu schema actuală. Procesul de unire este manual dar procesul de validare este automat.

În procesele definite de către acești autori schemele obiectuale se numesc *scheme virtuale*, clasele derivate se numesc *clase virtuale*, schema conceptuală se numește *schemă de bază*.

Metoda propusă de către Heiler și Zdonic [Heile 88], pentru definirea schemei externe presupune definirea a două mulțimi:

- mulțime de tipuri;
- mulțime de obiecte ca instanțe a acestor tipuri sau cu alte cuvinte definirea unui set de clase.

O schemă obiectuală este derivată prin aplicarea funcțiilor altei scheme în scopul de a obține noi definiri de scheme obiectuale. De aceea definirea unei scheme externe este de fapt definirea acestor funcții.

4.2.2 Clasificarea metodologiilor de definire a schemelor obiectuale

Metodologiile prezentate anterior pot fi grupate în trei grupe care iau în considerare relația dintre schema conceptuală şi schemele obiectuale definite:

- *Schemele obiectuale sunt subscheme ale schemei conceptuale* În acest grup de metodologii schema conceptuală conține toate clasele ce vor fi definite în schemele obiectuale. *Dacă schemele obiectuale conțin o clasă care nu a fost prevăzută în schema conceptuală aceasta trebuie să fie definită şi integrată în schema conceptuală.* Integrarea clasei asigură consistența tuturor schemelor externe cu schema conceptuală sau cu oricare altă schemă. Problema de bază care apare este aceea că schema conceptuală devine tot mai cmplexă pe măsură ce apar noi clase derivate. Schema conceptuală este folosită ca şi un dicționar de date în sensul că toate clasele incluse în schemele obiectuală vor fi integrate în dicționarul de date. În multe cazuri clasele derivate sunt integrate prin moştenire în schema conceptuală.
- *Schemele obiectuale nu conțin subscheme ale schemei conceptuale.* Schemele obiectuale pot conține clase care nu sunt incluse în schema conceptuală. Clasele noi sunt derivate direct sau indirect din schema conceptuală a claselor, dar ele nu sunt obligatoriu incluse în schema conceptuală. De aceea, schema conceptuală nu este afectată de definițiile schemei externe. *Ideea de bază a acestui grup de metodologii constă în faptul că schemele obiectuale sunt definite independent.*
- *Schema obiectuală este clasă a schemei conceptuale.* Acesta este un caz unic, unde fiecare schemă externă este implementată prin definirea unei clase noi, care conține un singur obiect, ce simulează comportarea tuturor claselor din noua schemă externă. Comparând cu schema conceptuală, acest mod de definire a schemei externe, înseamnă o schimbare a modului de definire a schemei externe precum şi a modului de lucru cu aceasta. Clasele care constitue din punct de vedere conceptual schema obiectuală, nu sunt clase care existau în schema conceptuală. Din acest punct de vedere, o astfel de metodologie poate fi considerată mai puțin importantă, comparativ cu cele precedente.

4.2.3 Generarea schemelor externe

Pe baza considerațiilor teoretice analizate anterior etapele necesare pentru a defini schema externă sunt:

- definirea şi integrarea claselor derivate cerute în dicționarul de date, ele putând fi direct integrate folosind relația derivată;
- selectarea claselor care vor face parte din schema externă. Clasele selectate pot fi transfomabile sau netransformabile;
- generarea schemei externe.

4.2.3.1 Modificarea claselor transformabile

În scopul de a avea o clasă transformabilă în ierarhia de clase, aceasta trebuie să sufere o serie de transformări, prin adăugarea de proprietăți sau prin ştergerea unor proprietăți deja existente. Modificările care pot avea loc într-o clasă transformabilă sunt reprezentate în figura 4.3. Considerăm clasa transformabilă clasa1 pe care vrem să o integrăm (fig 4.3.a). Dacă ea este subclasă a clasei clasa2 (fig 4.3 b) atunci ea moşteneşte proprietățile acesteia şi în plus îşi păstrează proprietățile sale proprii. În cazul în care clasa1 este superclasă pentru clasa2 atunci toate proprietățile care nu sunt definite şi pentru clasa2, trebuie eliminate din clasa1. (fig. 4.3.c)

Clasele transformabile moştenesc proprietățile superclaselor şi pierd acele proprietăți nedefinite pentru subclasele lor. (fig. 4.3.d). Dacă toate clasele care sunt subclase a unei clase transformabile au proprietăți comune şi aceste proprietăți sunt definite pentru toate obiectele din clasa transformabilă, atunci aceste proprietăți trebuie adăugate proprietăților clasei transformabile (proprietatea p5 in fig 4.3.d). Dacă o clasă transformabilă are mai multe superclase, atunci ea trebuie să moştenească toate proprietățile superclaselor (fig 4.3.e). Din această cauză transformările pe care le poate suferi o clasă

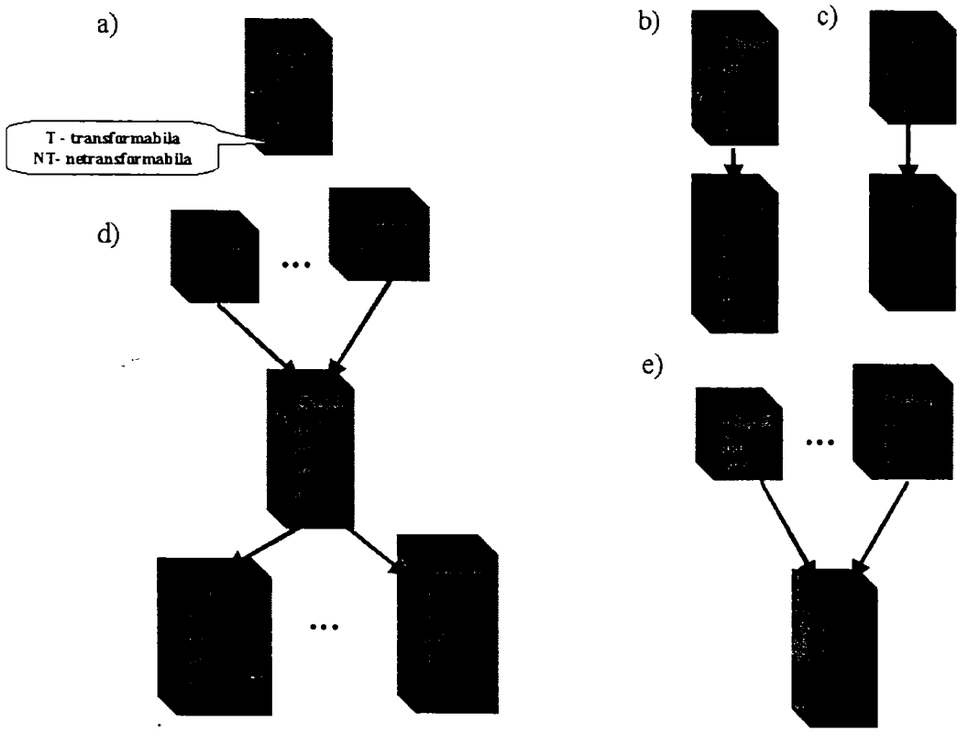


Fig 4.3 Reguli privind modificările care pot avea loc într-o clasă transformabilă

transformabilă când ea este integrată într-o ierarhie de clase pot fi:

- adăgare de proprietăți ale superclaselor care nu au fost definite în clasă;
- adăugare de proprietăți comune pentru toate subclasele şi pentru toate obiectele conținute în clase;
- eliminarea proprietăților nedefinite în subclase

4.2.3.2 Definirea ordinii de modificare a claselor transformabile în ierarhia de clase

Pentru definirea ordinii de modificare a proprietăților claselor transformabile sunt luate în considerare două aspecte:

- integrarea gradată a claselor transformabile în schema externă;
- toate clasele din schema sunt transformabile.

4.2.3.2.1 Integrarea gradată a claselor transformabile

Dacă privim ierarhia de clase formată exclusiv din clase netransformabile, clasele transformabile pot fi integrat gradat în ierarhia de clase. Din această cauză fiecare clasă transformabilă poate fi legată cu o clasă netransformabilă. O clasă transformabilă poate fi transformată în procesul de integrare, dar o dată integrată ea devine netransformabilă. Aşa cum se vede în figurile 3.b şi 3.c clasa transformabilă poate avea proprietăți ce pot fi eliminate sau adăugate acest lucru depinzând de legătura cu clasele netransformabile în ierarhia de clase. În general clasele transformabile care au fost deja integrate, pot numai să adauge proprietăți noi la alte clase transformabile (fig 4.3.b).

4.2.3.2.2 Schemă în care toate clasele sunt transformabile

În procesul de modificare al claselor transformabile pot să apară mai

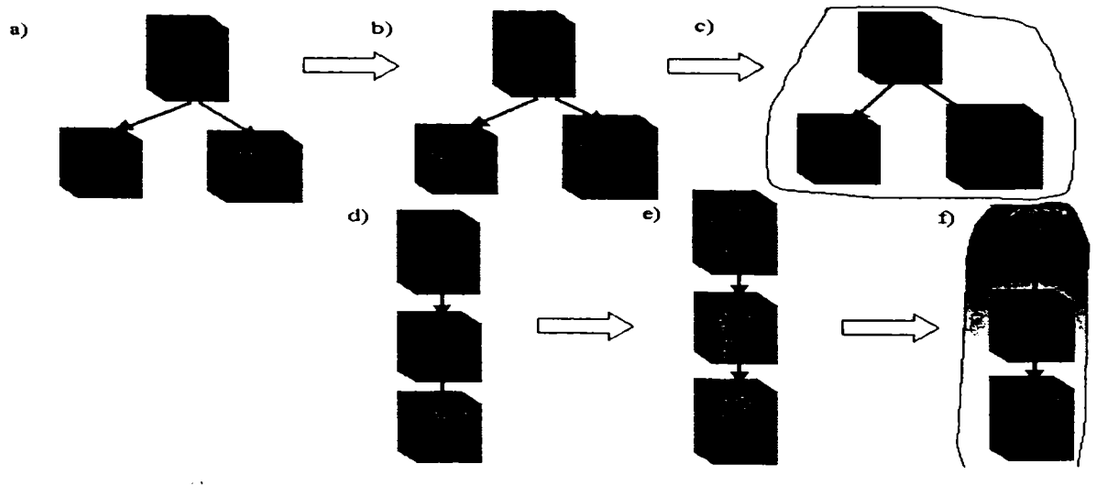


Fig 4.4 Procesul de modificare al claselor transformabile

multe aspecte. Dacă se caută păstrarea cât mai multor proprietăți posibile în transformarea claselor, primele transformări care trebuie făcute, sunt acelea prin care se adaugă proprietăți de la superclase sau pentru subclase.

În figura 4.4 sunt prezentate două cazuri de adăugare de proprietăți definite în superclasele unei clase transformabile. Figurile 4.4a şi 4.4b corespund primului caz care arată situația înainte şi după transformare. Figurile 4.4d şi 4.4e corespund cazului al doilea. În ambele situații proprietatea p3 este moştenită de către clasa3 de la clasa clasa1. Rezultatul transformărilor este prezentat în figurile 4.4c şi 4.4f. Aşa cum se poate vedea în ambele cazuri proprietatea p3 este eliminată din clasa1 dar ea rămâne definită în clasa clasa3. Dacă integrarea este făcută gradat proprietatea p3 nu va fi definită în clasa clasa3 pentru că ea a fost eliminată din clasa clasa1 la integrare.

În figura 4.5 este prezentată o transformare ce constă în adăugarea de proprietăți de la subclasele unei clase transformabile. Clasa2 şi clasa3 sunt clase netransformabile care au proprietatea p2 nedefinită în clasa1. În acest caz este posibil ca această proprietate să poată fi adăugată clasei transformabile, adică proprietatea p2 este adăugată la clasa1 fig. 4.5.b. Aceste noi proprietăți se pot propaga în structura de clase, p2 la clasa4 fig. 4.5.c. Dacă toate subclasele unei clase transformabile sunt clase transformabile orice proprietate definită în una dintre subclase poate fi adăugată la clasa transformabilă, în cazul în care proprietatea apare la toate obiectele.

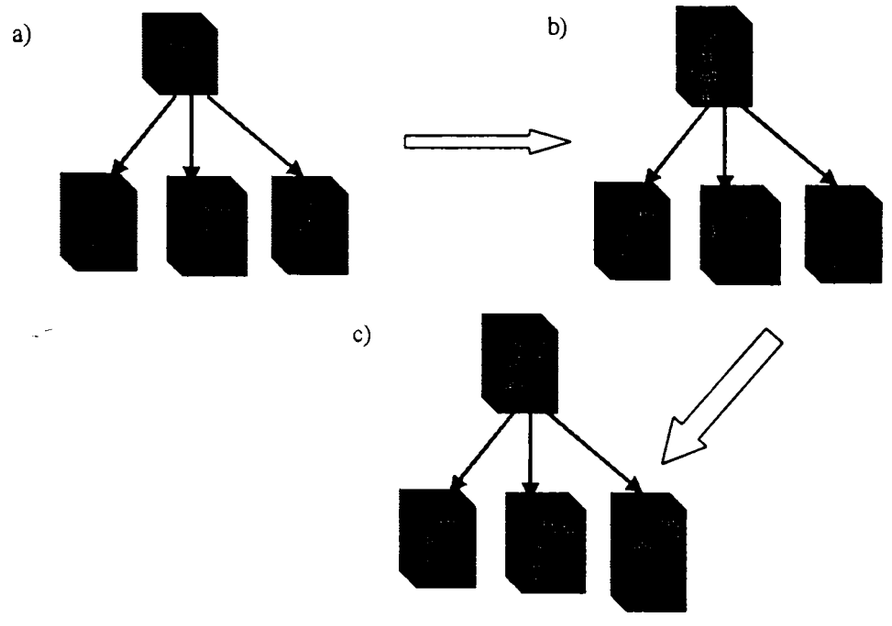


Fig 4.5 Adăugare de proprietăți de la subclasele unei clase transformabile

O dată ce toate proprietățile posibile ale superclaselor şi subclaselor au fost adăugate la clasa transformabilă, trebuie eliminate toate proprietățile clasei transformabile nedefinite în subclase aşa cum se arată în figurile 4.4c şi 4.4f

4.2.3.3 Moştenirea între clasele transformabile şi netransformabile

O dată ce au fost selectate clasele transformabile şi cele netransformabile în scopul de a modifica clasa transformabilă s-au obținut şi relațiile de moştenire între clase. Relațiile de moştenire determină calea prin care se vor face modificările clasei transformabile.

Există posibilitatea de a defini moştenirea manual în momentul în care este făcută selecția sau automat de către sistem

4.2.3.3.1 Caracteristici ale ierarhiei de clase

Pentru a obține caracteristicile ierarhiei de clase există două soluții:

- obținerea ierarhiei închise de clase, legată de relațiile de moştenire, adăugând la setul de clase selectate, clasele de care avem nevoie;
- obținerea tuturor relațiilor de moştenire din setul de clase selectate.

4.2.3.3.2 Subsumarea relațiilor

Mulțimea proprietăților unei clase numită intensie sau grad, este definită ca reuniunea unui set de atribute şi a unui set de metode. *Intensia unei clase, mai poate fi definită ca reuniunea unui număr de proprietăți ale clasei. Extensia, este numărul de obiecte (instanțe) al unei clase.* Intensia şi extensia sunt definite ca operații noi în schema externă. Extensia unei clase poate fi definită în scopul de a cataloga o clasă, ca o superclasa pentru două clase existente

În cazul claselor netransformabile intensia (gradul) şi extensia sunt cunoscute dinainte. Cu alte cuvinte clasele transformabile au extensia stabilită dinainte dar nu şi intensia. Intensia poate fi modificată în scopul de a fi adaptat funcție de poziția clasei în ierarhia de clase. De aceea, relațiile dintre clase netransformabile trebuie legate de extensia şi intensia lor, iar relațiile dintre clasele transformabile şi cele netransformabile trebuie legate de intensia lor.

Relațiile de moştenire între clase pot fi obținute, folosind funcția de *subsumare* bazată pe relațiile existente între clase, în dicționarul de clase. Această funcție se referă la intensia şi extensia claselor. În cazul claselor transformabile este luată în considerare numai extensia. *Clasa c1 subsumă clasa c2 dacă şi numai dacă c1 poate fi definită ca o superclasă a lui c2. Aceasta înseamnă că setul de obiecte a lui c1 conține întodeauna setul de obiecte a lui c2.*

Funcția de subsumare (*subsumes()*) a fost studiată în [Runde 92-2]. În cazul claselor netransformabile relațiile obținute între clase depind exclusiv de funcția de subsumare utilizată. În altă ordine de idei în cazul claselor transformabile există multiple posibilități de a modifica intensia unei clase. În [Berti 92] funcția de subsumare este folosită în scopul verificării corectitudinii schemei externe elaborată manual.

4.2.3.3.3 Subsumarea claselor

Dacă extensia unei clase transformabile clasa1, este subsumată de extensia unei alte clase, dar ea nu subsumează nici o altă clasă în cadrul ierarhiei

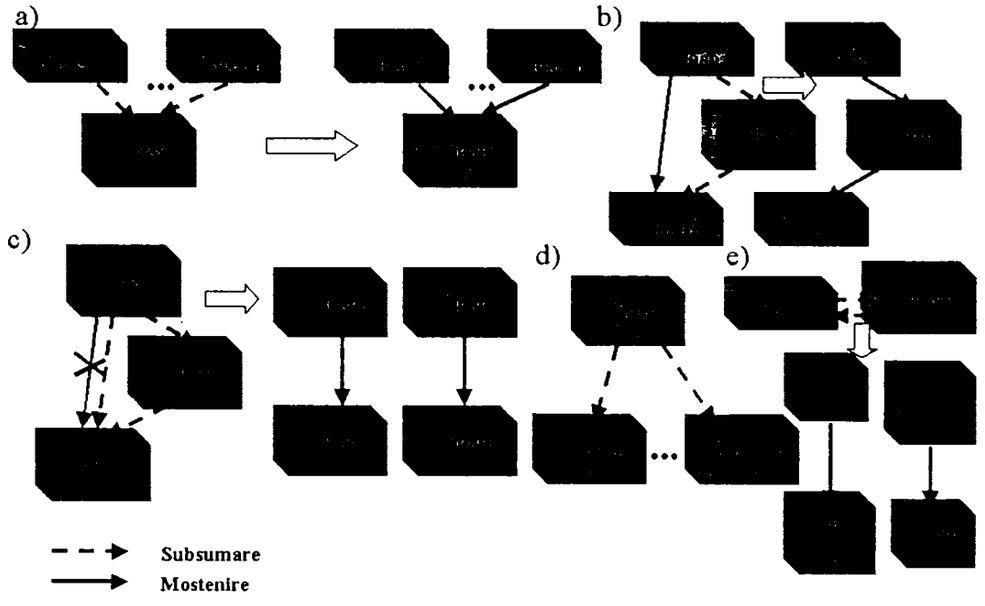


Fig 4.6 Subsumarea claselor

de clase, putem spune că această clasă transformabilă este o subclasă a tuturor claselor care o subsumează (fig 4.6 a). De aceea, subsumarea extensiilor relațiilor poate fi transformată direct în relații de moştenire. Altfel dacă o clasă transformabilă clasa1, este subsumată de alte clase şi ea la rândul său subsumează alte clase dar toate clasele subsumate de clasa1 sunt subclase ale claselor care o subsumează atunci clasa clasa1 poate fi definită ca subclasă a tuturor claselor care o subsumează şi o superclasă pentru toate clasele pe care le subsumează (fig 4.6 b).

În cazurile prezentate în figura 4.6.a şi 4.6.b relațiile de moştenire între clase şi restul claselor pot fi definite direct, ca subsumarea extensiei relațiilor. În alte situații în transformarea sumării extensiei relațiilor între clasele transformabile şi alte clase legate prin moştenire există diverse posibilități. De exemplu în cazul prezentat în figura 4.6.c clasa transformabilă clasa1 este

subsumată de clasa netransfomabilă clasa3; de asemenea clasa1 subsumează clasa netransformabilă clasa2, de aceea clasa2 nu este o subclasă a lui clasa3. Astfel transformând relațiile de subsumare în relații de moştenire, există posibilitatea ca clasa1 să fie definită ca subclasă a lui clasa3 sau să definim clasa1 ca şi superclasă a lui clasa2. Niciodată nu este posibil să le definim pe amândouă, pentru că clasa3 nu este superclasă a lui clasa2. O altă posibilitate, este de a considera clasa transformabilă clasa1 de două ori în ierarhia de clase şi de a genera două clase diferite în cadrul schemei, una pentru relația cu clasa3 şi una pentru relația cu clasa2. Dacă o clasă transformabilă clasa1, subsumează în extensie un set de clase netransformabile (fig 4.6.d), care nu au nici o proprietate în comun şi clasa clasa1 este definită ca o superclasă a lor, rezultatul transformărilor poate fi o clasă fără proprietăți, dar acesta nu este în general un rezultat acceptabil. De aceea o clasă transformabilă poate fi definită ca o superclasă numai în cazul în care subsumează una sau mai multe clase. Dacă clasa transformabilă şi o altă clasă selectată au aceeaşi extensie (fig 4.6.e, clase subsumate în extensie unele cu altele), clasa transformabilă poate fi definită ca o superclasă, tot atât de bine ca şi o subclasă a altei clase, în ierarhia de clase şi ambele posibilități pot fi valide, chiar dacă există şi posibilitatea ca ambele cazuri să nu apară. De aceea unul din cazuri trebuie selectat. O altă posibilitate este aceea că putem avea clasa clasa1, de două ori în ierarhia de clase, o dată ca subclasă şi o dată ca superclasă a unei clase, care are aceeaşi extensie şi generează după transformare, două clase diferite din clasa clasa1. Dacă atât clasa clasa1 cât şi clasa clasa2 sunt clase transformabile, ele pot devenii aceaşi clasă după transformare. În scopul de a evita situația în care există mai multe posibilități de definire a relației de moştenire pot fi luate în considerare următoarele soluții:

- evitarea claselor în care pot apare astfel de situații;
- analizarea tuturor relațiilor de subsumare între clase transformabile şi alte clase şi alegerea relațiilor de moştenire dorite;
- definirea criteriilor pentru selecția automată a relațiilor de subsumare care vor fi transformate în relații de moştenire;
- prezența mai multor clase transformabile în cadrul ierarhiei de clase şi generarea altor clase din fiecare dintre ele.

4.2.3.4 Proprietatea "închidere" a unei relații

O ierarhie de clase este închisă printr-o proprietate specifică a relației dacă şi numai dacă pentru toate clasele incluse în ierarhia de clase, există o proprietate de legătură între ele, cu care sunt incluse în ierarhia de clase.

4.2.3.4.1 Cerințele proprietăților clasei

O clasă netransformabilă poate referenția alte clase incluse în mulțimea claselor selectate. Aceste clase pot fi transformabile sau netransformabile. O clasă netransformabilă poate face referire la unele proprietăți sau metode definite în clasele la care face referire. Aceste proprietăți pot de asemenea referi alte proprietăți definite în clase dar numai în mod indirect cele definite în clase netransformabile. Dacă o proprietate este referențiată direct sau indirect de către o clasă netransformabilă şi ea este definită într-o clasă transformabilă, atunci clasa rezultată prin modificarea unei clase transformabile va avea inclusă proprietatea referențiată. De aceea *proprietățile claselor transformabile pot fi referite dacă ele sunt referențiate de către o clasă netransformabilă şi fără să se facă referire la ele în celelalte cazuri.*

Fiecare proprietate a clasei este legată de un set de proprietăți din aceeaşi clasă sau din clase diferite. Aceasta lucru este necesar în scopul de a avea proprietatea definită. Toate proprietățile unei clase netransfomabile sunt proprietăți cerute. Dacă o proprietate cerută a unei clase transformabile face referință la o clasă care inițial nu a fost selectată, pentru a face parte din schema externă, poate fi inclusă în setul de clase care va face parte din schema externă. Dacă o proprietate necerută a unei clase transformabile, este eliminată când se face modificarea clasei, toate proprietățile acestei clase sau a altei clase transformabile, care face referință la clasa eliminată, sunt eliminate. Aceste proprietăți vor fi nereferențiate. Adevărul privind includerea acestor clase în mulțimea de clase selectate, este că unele dintre aceste proprietăți sunt cerute direct sau indirect de către clasele netransformabile. De aceea dacă ele sunt considerate clase transformabile, toate cerințele despre ele vor fi îndeplinite.

4.2.3.4.2 Clase netransformabile care referă clase transformabile

Când o clasă transformabilă este modificată, o nouă clasă este generată. De aceea dacă o clasă netransformabilă referențiază clase transformabile şi clasele transformabile sunt înlocuite de clase noi în schema externă, la clasele netransformabile se va modifica doar referința la clasele transformabile ce au fost modificate. Dacă s-a modificat referențierea la alte clase netransformabile, atunci vor fi generate noi clase care să înlocuiască clasele netransformabile în schema externă. În procesul descris mai sus clasele netransformabile trebuie să fie înlocuite de clase noi în schema externă în scopul de a schimba referințele la clasele noi generate. În scopul de a permite acest lucru, simplificarea procesului de generare al schemei externe, constă în faptul că orice clasă referențiată de o clasă netransformabilă trebuie calificată ca netransformabilă. De aceea *clasele transformabile referențiate de clase netransformabile, sunt catalogate ca*

netransformabile, iar clasele referențiate de clasele netransformabile și care nu au fost incluse inițial în setul de clase selectate pentru a face parte din schema externă, sunt de asemenea catalogate ca netransformabile.

4.2.3.4.3 Transformarea referințelor la clase

Dacă o clasă netransformabilă sau o proprietate a unei clase transformabile, face referință la o altă clasă neinclusă în mulțimea de clase ce vor forma schema externă, clasele referite pot să nu fie incluse în schema externă. Dacă o clasă derivată este referențiată de către o clasă originală, dacă această clasă originală este inclusă în setul de clase selectate și dacă are incluse toate proprietățile cerute, atunci clasa care referențiază poate fi înlocuită în schema externă prin clasa derivată.

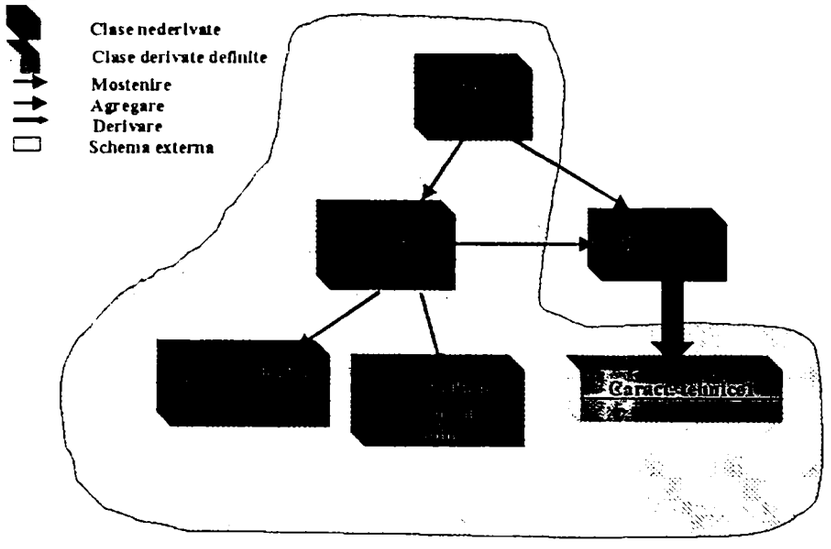


Fig 4.7 Exemplu de schemă externă

În figura 4.7 este arătat un exemplu, în care va fi definită schema externă în care se va înlocui clasa "Caract. Tehnice" cu noua clasă "Caract. Tehnice1" care nu are inclusă proprietatea "desen". Cu comportamentul descris anterior clasa definită explicit este "Caract. Tehnice1". Restul claselor vor fi automat adaptate la noua situație, rezultând schema externă dorită, prezentată în figura 4.8.

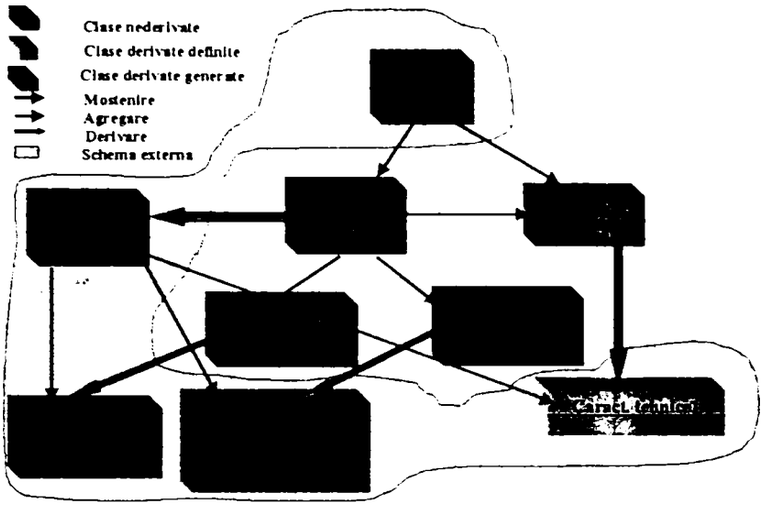


Fig 4.8 Schemă externă

4.2.3.5 Alternative în procesul de definire al schemelor externe

Alternativele care pot apare în procesul de definire al schemei externe pot fi următoarele:

1. Organizarea schemei externe:
   - Schemele externe pot fi închise în acord cu relațiile de moştenire;
   - Nu este obligatoriu ca schemele externe să fie închise în acord cu relațiile de moştenire
2. Calificarea claselor selectate:
   - Claselor selectate care compun schema externă pot fi calificate ca şi clase transformabile sau netransformabile;
   - Clasele selectate pentru a forma schema externă, nu pot fi transformate, de aceea toate clasele trebuie să fie netransformabile.
3. Scopul pentru care sunt transformate clasele

Transformarea claselor se face funcție de scopul care se doreşte atins. În acest sens putem avea:

- În fiecare moment în ierarhia de clase a schemei externe există o singură clasă care poate fi transformată;
- Prima dată clasa este integrată şi apoi transformată funcție de situația în care se găseşte.

4.3 Concepte privind definirea claselor derivate

Clasa derivată este o clasă care este definită din clase existente, folosind diverse criterii particulare. Clasele nederivate sunt definite la implementarea schemei conceptuale. Clasele derivate sunt definite pe parcursul existenței bazei

de date, în scopul de a fi incluse în scheme externe sau în scheme conceptuale. Orice clasă derivată poate fi folosită ca şi o clasă nederivată.

4.3.1 Integrarea claselor derivate în schema obiectuală

Integrarea claselor derivate este privită din două puncte de vedere diferite:

- din punct de vedere al existenței claselor în schema conceptuală;
- în scopul de a forma o schemă externă.

În definirea schemelor externe, din punct de vedere al paradigmei obiectuale, *integrarea înseamnă folosirea relației de moştenire*. În multe cazuri acest lucru nu este respectat. Din alt punct de vedere obiectivul integrării noilor clase derivate cu restul claselor existente în schema obiect prezintă două aspecte şi anume:

1. menține relațiile dintre clasele derivate şi cele nederivate pentru a păstra consistența schemei definite;
2. integrarea claselor este folosită pentru modelarea datelor.

Pornind de la schema obiect prezentată în figura 4.9 se poate arăta integrarea unei clase derivate în ierarhia de clase. Este definită noua clasă "Freze cilindro frontale fără plăcuțe din carburi metalice" din clasa "Freze cilindro frontale", care nu are proprietatea cod_plăcuță şi care conține obiecte freze cilindro frontale ce nu au plăcuțe. Dacă noua clasă trebuie integrată în ierarhia de clase aceasta înseamnă că ea nu este o subclasă a clasei inițiale "Freze cilindro frontale", pentru că nu are proprietatea cod_plăcuță. Ea nu poate fi o superclasă pentru "Freze cilindro frontale" pentru că obiectele sale sunt o submulțime a clasei inițiale "Freze cilindro frontale". Aceasta înseamnă că, clasa "Freze cilindro frontale fără plăcuțe din carburi metalice" este clasă derivată din clasa "Freze cilindro frontale".

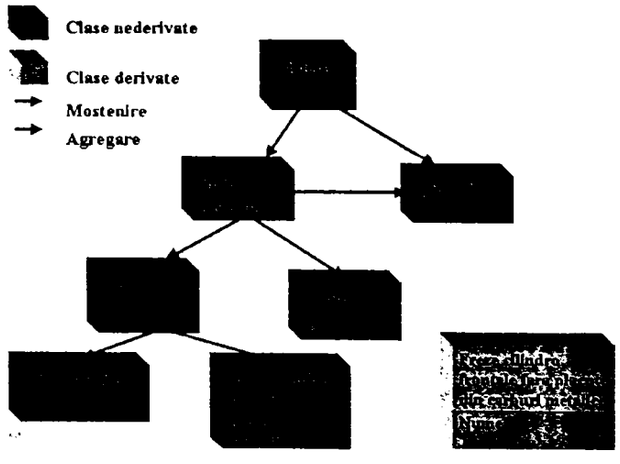


Fig 4.9 Integrarea clasei derivate în ierarhia de clase

Integrarea claselor derivate în schema externă se poate face folosind diverse relații existente între clase sau relația de moştenire

4.3.1.1 Integrarea folosind relații diferite între clase altele decât relația de moştenire

Acest concept ignoră rezultatele determinării relațiilor dintre clasele derivate şi alte clase, folosind tipuri de relații diverse.

4.3.1.1.1 Derivarea relației

Fiecare clasă derivată este legată cu clasa din care a fost definită. Folosind o relație derivată se defineşte calea prin care clasa derivată este obținută independent. În figura 4.10 clasa "Freze cilindro frontale fără plăcuțe din carburi metalice", este direct integrată folosind relația derivată.

Această relație este numită *vedere derivată* în [Berti 92], [Guerr 97] iar în [Kim 95] clasele derivate sunt legate cu clasele din care ele au fost definite, prin relații derivate din relație. În [Kim 95], mulțimea claselor legate prin acest tip de relație se numeşte *ierarhia de vederi derivate*, iar clasele derivate sunt legate cu celelalte clase printr-o relație de tip "derivată din" (derived from). În [Naja 95] acest tip de relație se numeşte "este derivată din" (is derived from), restricțiile fiind mai puternice decât în alte tipuri de relații. Extensia a două clase legate prin acest tip de relație trebuie să fie de acelaşi tip, singura diferență între clase, poate consta în setul de atribute. Din această cauză acest tip de relație este folosit doar în dicționarele de date.

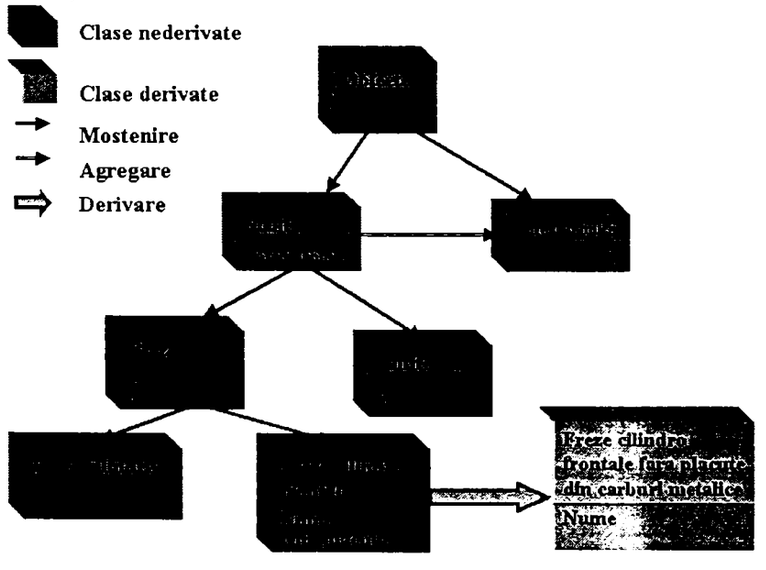


Fig 4.10 Derivarea relației

4.3.1.1.2 Cluster de clasă

În scopul de a aprofunda problema integrării claselor derivate cu alte clase, soluția propusă în [Heuer 91] constă în definirea clusterilor de clasă: *clusterul este o clasă de bază şi câteva clase derivate având acelaşi set de obiecte ca şi clasa de bază*. În interiorul unui cluster sunt considerate două tipuri de ierarhii:

- una de tipuri;
- una de moşteniri.

Fiecare clasă derivată este clasificată în ierarhii ce aparțin clusterului respectiv. În exemplul considerat clasa derivată "Freze cilindro frontale fără plăcuțe din carburi metalice" este inclusă în clusterul corespunzător clasei de bază, "Freze cilindro frontale", din care este definită.

4.3.1.1.3 Relația "May_be"

Relația May_be a fost propusă de către [Santo 94], în scopul de a integra clasele derivate în scheme externe (aceasta înseamnă o extensie a paradigmei orientate obiect). Relația este denumită *may_be* pentru a se distinge de relațiile uzuale de moştenire, numite *is_a*. În acest sens, o instanță a unei clase de bază poate fi o instanță ce corespunde unei clase derivate. În exemplul prezentat clasa derivată "Freze cilindro frontale fără plăcuțe din carburi metalice" este definită folosind semantica obiectuală de aceea rezultatul obținut poate fi de acelaşi tip cu cel prezentat în figura 4 pentru relația derivată.

4.3.1.2 Integrarea claselor derivate folosind moştenirea

Integrarea clasei derivate folosind moştenirea poate fi definită în mai multe moduri şi anume:

- prin legătura cu clasa de bază;
- prin definirea unor subclase ale claselor de obiecte;
- definirea tuturor relațiilor posibile între clasele derivate şi celelalte clase;
- relații definite explicit între clasele derivate.

4.3.1.2.1 Legătură clasei derivate numai cu clasa de bază

Prin această soluție fiecare clasă derivată este legată prin moştenire numai cu clasa sursă. Problema care se pune, este de a vedea ce se întâmplă în cazul în care clasa derivată nu este legată direct prin moştenire cu clasa sau clasele sursă.

Această problemă permite evitarea definiri acelor clase derivate, care sunt în legătură directă cu clasele sursă. Din această cauză putem avea rezultate foarte bune folosind ierarhia parțială a claselor. Ca o concluzie se poate spune că pentru fiecare definiție de clasă derivată, relația dintre clasele participante este definită.

4.3.1.2.2 Clasa derivată definită ca subclasă a unei clase de obiecte

O posibilitate este de a defini o nouă clasă derivată, ca o subclasă a unei clase de obiecte (fig 4.11). Acest concept, ignoră clasificarea, de aceea rezultatul este o structură plată care nu prezintă avantajele moştenirii şi a paradigmei obiectuale. Această soluție este adoptată în sistemul dezvoltat de către Kim [66]. Din figură se observă că definirea clasei "Freze cilindro frontale fără plăcuțe din carburi metalice" cu semantica obiectuală nu permite definirea altor clase derivate.

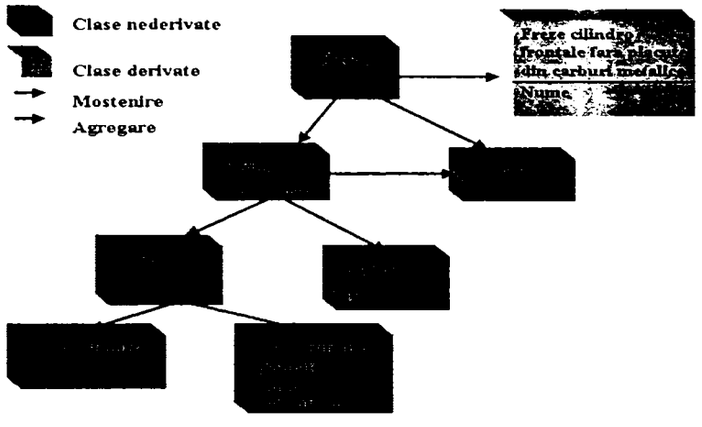


Fig 4.11 Clasa derivată definită ca subclasă a unei clase de obiecte

4.3.1.2.3 Relații definite explicit între clasele derivate definite şi clasele existente

Prin acest concept proiectantului de aplicație îi este cerut să specifice explicit moştenirea relațiilor între clasele derivate definite şi clasele existente. Pentru verificarea corecturilor ce trebuie aduse pentru funcționarea normală a unei astfel de soluții, este necesară realizarea unui sistem care să prevadă verificarea automată a ierarhiilor de clase definite.
În figura 4.12 clasa derivată "Freze cilindro frontale fara placute din carburi metalice" este definită numai ca o subclasă a clasei "Freze". Ea nu poate fi legată direct cu clasa "Freze cilindro frontale" folosind o relație de moştenire. Acest mod de definire a relațiilor între clasele selectate pentru a compune

schemele obiectuală, este analizat în [Guerr 97], [Dayal 89]. Corectarea relațiilor de moştenire este asigurată de către sistem.

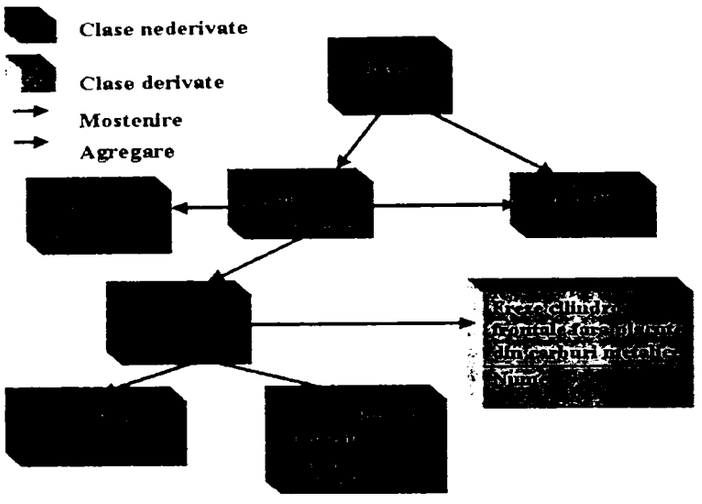


Fig 4.12 Relații definite explicit între clasele derivate şi clasele existente

4.3.1.2.4 Definirea tuturor relațiilor posibile

Integrarea manuală poate fi considerată ca o alternativă la integrarea automată. Astfel sunt obținute explicit toate relațiile de moştenire existente între clasele derivate şi restul de clase. Pentru exemplul considerat, rezultatul obținut poate fi cel din figura 12. Schema obiect rezultată poate părea incompletă: clasele "Freze cilindro frontale" şi "Freze cilindro frontale fără plăcuțe din carburi metalice" au obiecte şi proprietăți comune şi ceea ce este semnificativ este faptul că, însuşi clasa "Freze" şi componentele sale, nu este reprezentată explicit în ierarhia de clase.

În metologia Rundensteiner [Runde 92-1] procesul de integrare a clasei derivate rezolvă această problemă. În scopul de a integra clase derivate prin moştenire într-o schemă obiect, aceasta cere ca fiecare pereche de clase să aibă proprietăți în comun. Una dintre ele este o superclasă care conține toate proprietățile comune ale ambelor clase şi care trebuie introdusă în schema obiect. Această proprietate se numeşte *închiderea moştenirii* a schemei obiect. O astfel de schemă este prezentată în figura 4.13. În scopul de a integra clasa " Freze cilindro frontale fără plăcuțe din carburi metalice " toate clasele cer menținerea schemei obiect în acord cu cele enunțate automat la generarea clasei "Freze cilindro frontale"

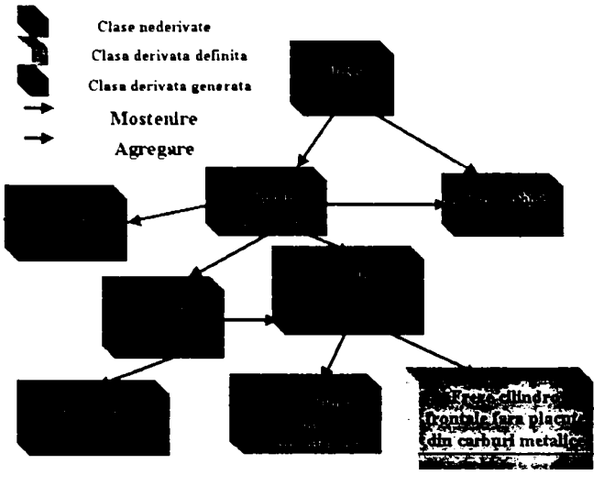


Fig 4.13 Schemă externă care satisface închiderea moştenirii

Pentru a arăta acest proces în detaliu, în figura 4.14 este prezentat un alt exemplu de integrare. În acest caz clasa " Freze cilindro frontale fără plăcuțe din carburi metalice " este definită fără proprietatea cod_plăcuță, dar cu noua proprietate "Desen" care returnează desenul sculei. Această proprietate este de asemenea definită în clasa "Caract. Tehnice". În acest caz problema moştenirii între "Freze cilindrice" şi " Freze cilindro frontale fără plăcuțe din carburi metalice " depinde de tipul claselor. Este generată o nouă clasă "Freze cilindro frontale 1" care conține un set de obiecte diferite față de cele două clase. De asemenea este generată o altă clasă cu numele "Desene". Această clasă nouă conține proprietățile comune claselor "Caract. Tehnice" şi " Freze cilindro frontale fără plăcuțe din carburi metalice". Astfel aceste proprietăți pot fi moştenite de ambele clase.
Esența acestei soluții este de a crea clase intermediare.

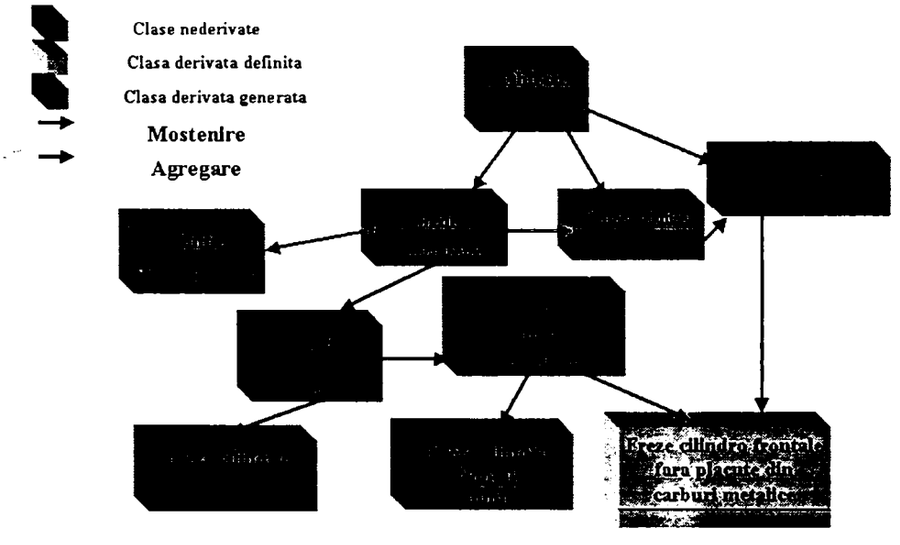


Fig 4.14 Integrarea clasei derivate, prin definirea tuturor relațiilor existente, între clasa derivată şi clasa de bază

4.3.1.3 Integrarea claselor derivate în schemele unui OODB

Considerăm schema conceptuală din fig 4.15 şi clasa derivată "Freze cilindro frontale fără plăcuţe din carburi metalice", definită din clasa "Freze cilindro frontale", care nu are proprietatea cod_plăcuţă şi are ca obiecte numai acele freze care nu conţin plăcuţe. Va fi definită o schemă externă cu aceeaşi structură ca şi schema conceptuală dar cu noua clasă "Freze cilindro frontale fără

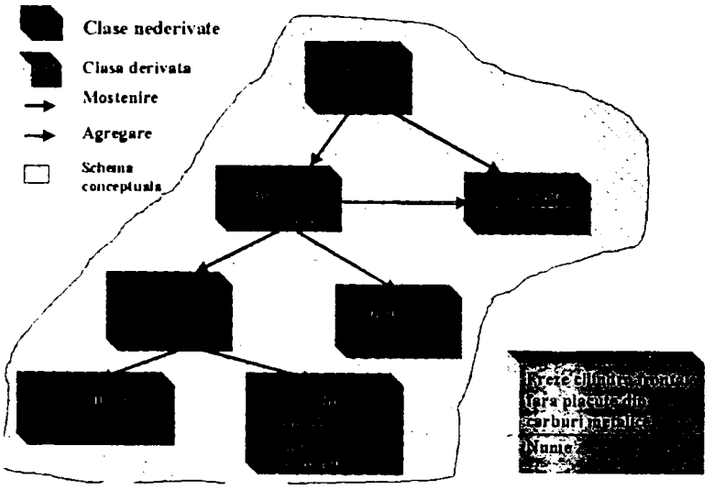


Fig 4.15 Schemă externă

plăcuţe din carburi metalice" înlocuind clasa "Freze cilindro frontale".

4.3.1.3.1 Integrarea claselor derivate în dicţionarul de date

Clasele derivate nu trebuie incluse în schema conceptuală, înainte de a fi incluse în schema externă. Datorită acestui fapt ele trebuie definite şi incluse în dicţionarul de date.

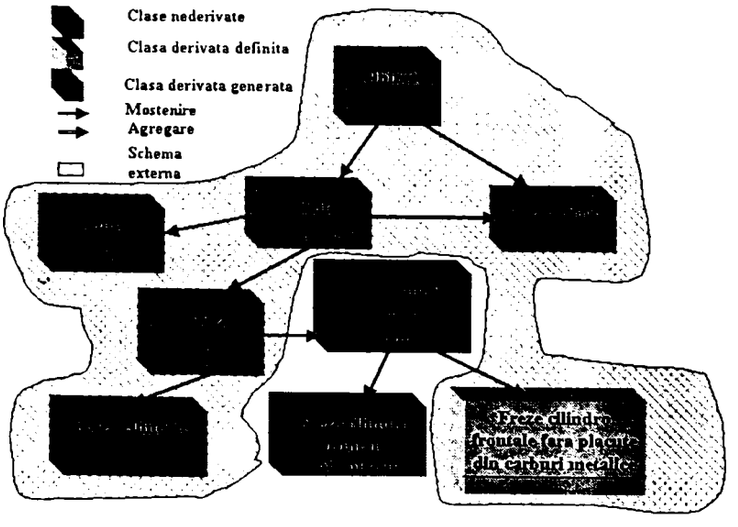


Fig 4.16 Selecţia unei mulţimi de clase

De multe ori legătura care se bazează pe moştenire, dintre clasa de bază şi clasa derivată nu se face direct. De aceea în scopul de a avea o legătură explicită bazată pe moştenire este necesară folosirea claselor intermediare [Runde 92-2], [Runde 92-3], aşa cum a fost arătat în figura 4.13. Figura 4.16 arată selecția unei mulțimi de clase care vor compune schema externă considerând exemplul prezentat. Pentru a integra clasa derivata "Freze cilindro frontale fără plăcuțe din

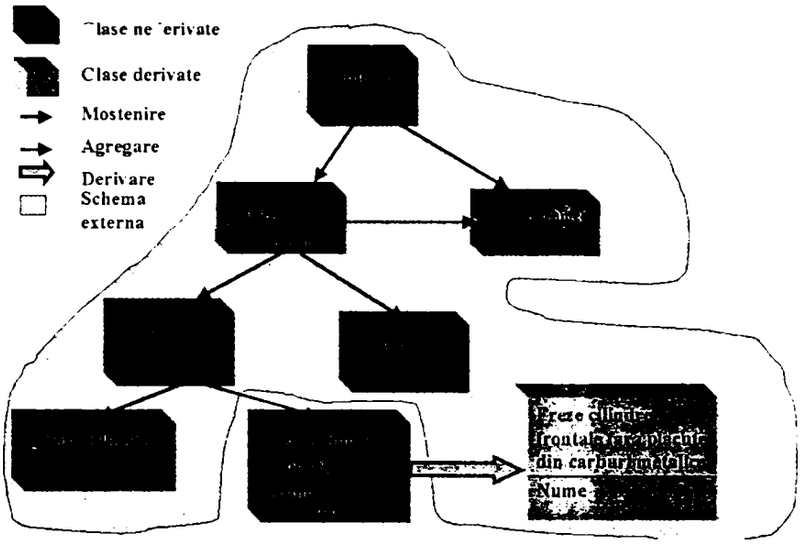


Fig 4.17 Folosirea relației de tip cluster de clasă

carburi metalice" a fost generată automat clasa "Freze cilindro frontale 1" şi adăugată la dicționarul de clase, dar această clasă, nu va fi inclusă în schema externă. Această clasă reprezintă exclusiv relația de moştenire dintre clasa originală de bază "Freze cilindro frontale" şi clasa derivată "Freze cilindro frontale fără plăcuțe din carburi metalice". Oricum schema externă va include numai una dintre ele.

O altă soluție este aceea de a folosi relația derivată de tip cluster de clase aşa cum este prezentată în figura 4.17. Noua clasă derivată, este integrată direct în dicționarul de date prin relația derivată, fără a mai fi necesară generarea unei clase adiționale.

4.3.1.3.2 Integrarea claselor derivate în schema externă

Când clase noi derivate sunt integrate în dicționarul de date aşa cum este arătat în figurile 4.16 şi 4.17, o mulțime de clase pot fi selectate pentru a forma o schemă externă. În cazul integrării claselor derivate prin moştenire în dicționarul de date aşa cum se vede în figura 4.16, *două clase sunt legate prin moştenire în schema externă dacă şi numai dacă ele sunt legate prin moştenire direct sau indirect în dicționarul de date.* Procesul de integrare în dicționarul de date constă în definirea explicită a tuturor relațiilor de moştenire dintre clase. Dacă

integrarea în dicționarul de date se face folosind relații derivate, pot fi descoperite unele relații de moștenire noi între clase. Dacă două clase incluse în schema externă sunt legate prin moștenire în dicționarul de date atunci ele pot fi de asemenea legate prin moștenire în schema externă. De aceea pot exista clase care nu sunt legate prin relația de moștenire în dicționarul de date, dar relația de moștenire între clase să existe. În exemplul din figura 4.17 a fost obținută relația de moștenire între clasele "Freze cilindro frontale fără plăcuțe din carburi metalice" şi "Freze".

4.3.1.3.3 Diferența între integrarea claselor în dicționarul de date şi schema externă

Integrarea este un proces care se poate desfăşura în două moduri: în dicționarul de date şi în schema externă. Pentru a arăta clar diferența dintre cele două moduri de integrare prezentăm următorul exemplu. Pornind de la schema conceptuală din figura 4.15 trebuie definită o schemă externă cu aceeaşi structură dar în loc de proprietatea Caract. Tehnice, în clasele "Freze", "Freze cilindrice", "Freze cilindro frontale", avem proprietatea Desen definită în clasa Caract. tehnice. De asemenea în clasa Freze cilindrice nu există proprietatea cod_placută.

Definirea schemei externe este arătată în figura 4.18, dacă clasele derivate sunt integrate folosind algoritmul lui Rundensteiner [Runde 92-4]. Prima dată clasele "Freze cilindrice" şi "Freze cilindro frontale fără plăcuțe din carburi metalice", sunt definite funcție de cerințe. Apoi ele sunt integrate în schema conceptuală generând prin această operație clase adiționale. Nu este necesară definirea unei noi clase "Freze" pentru că ea este generată prin algoritmul de generare. În final sunt selectate clasele pentru a obține schema externă.

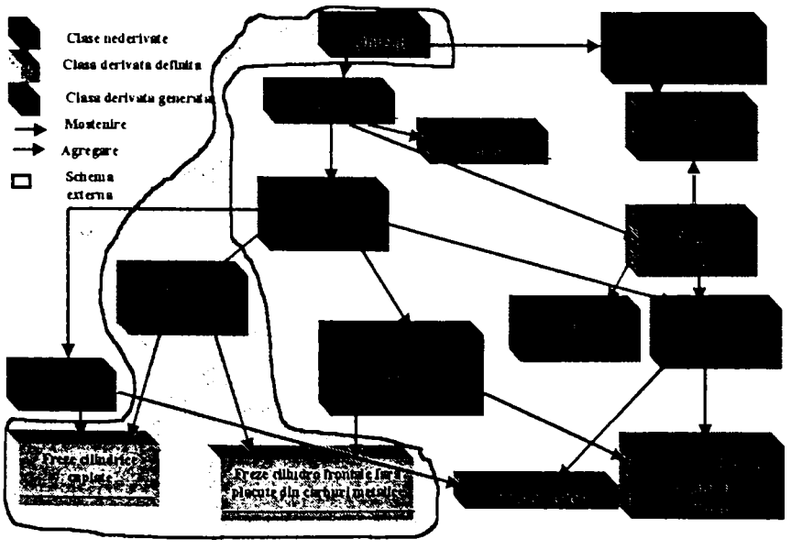


Fig 4.18 Definirea schemei externe folosind algoritmul lui Rundensteiner

Dacă integrarea claselor derivate în dicționar, este făcută folosind relația de derivare acest lucru este arătat în figura 4.19. În acest caz procesul de integrare nu va genera clase adiționale. Din această cauză toate clasele trebuie să

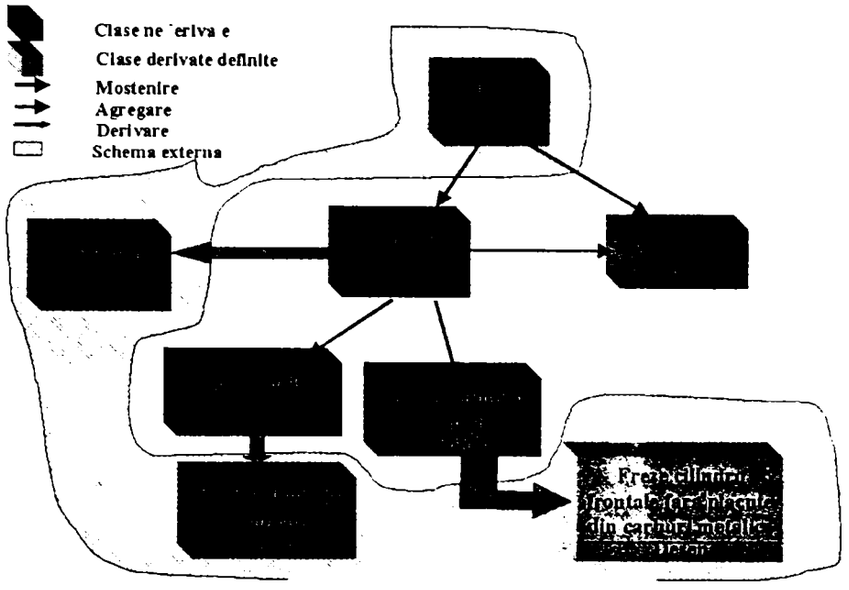


Fig 4.19 Integrarea claselor derivate în dicțîonarul de date

fie definite explicit şi dicționarul de date să conțină numai clasele strict necesare în scopul de a defini schema externă.

Dacă se integrează în dicționarul de date clase noi derivate prin moştenire atunci:

- un mare număr de clase sunt generate automat şi numai câteva dintre ele sunt folosite pentru definirea schemei externe adică cu alte cuvinte toate clasele din dicționarul de clase au fost luate în considerare în procesul de integrare;
- unele dintre clasele cerute pentru a fi integrate în schema externă nu au fost definite explicit în procesul de integrare;
- relațiile de moştenire între clasele selectate să compună schema externă pot fi obținute direct pentru că relațiile de moştenire între clase au fost definite inițial în dicționarul de clase, din această cauză *două clase pot fi legate prin moştenire în schema externă numai dacă relația a fost definită inițial în dicționarul de clase.*

Din alt punct de vedere, dacă clasele derivate, sunt integrate în dicționarul de date folosind relația derivată aşa cum se poate vedea în figura 19 atunci:

- nu sunt generate automat clase în procesul de integrare a claselor derivate în dicționarul de date;
- toate cerințele claselor sunt definite explicit;
- efortul de integrare este mai complex, în sensul că relații de moştenire nedefinite în dicționarul de date sunt descoperite în timpul procesului,

dar sunt luate în considerare doar acele clase, care vor fi folosite în cadrul schemei externe.

4.3.2 Tipuri de clase existente în schema externă

În scopul de a simplifica procesului de definire al schemei externe, trebuie reduse numărul de clase care trebuie definite explicit. O soluție, este de a defini

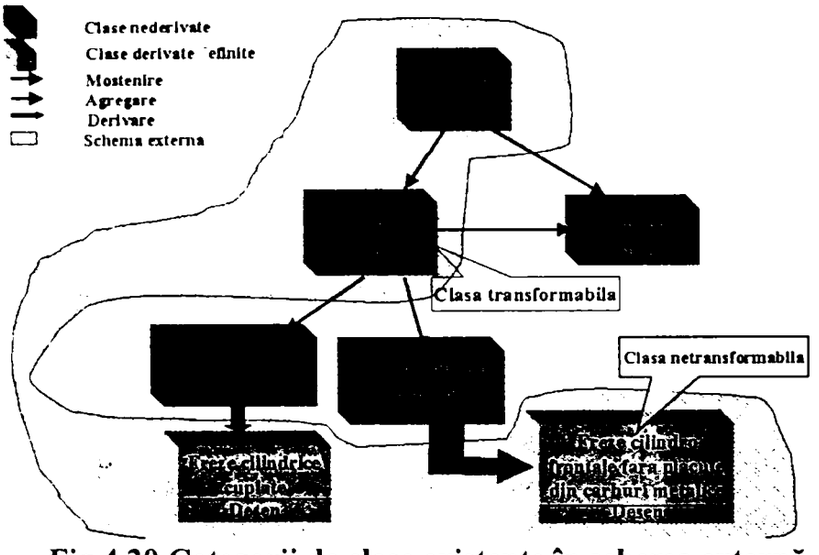


Fig 4.20 Categorii de clase existente în schema externă

categorii diferite de clase, atunci când este făcută selecția claselor care trebuie să compună schema externă.

Se disting două categorii de clase: transformabile şi netransfomabile. Clasele netransformabile sunt adăugate direct la schema externă. În exemplul prezentat toate clasele erau considerate netransfonmabile. Clasele transformabile pot fi înlocuite, în schema externă de clase derivate dacă este necesar. Aceste noi clase sunt rezultatul modificării claselor originale prin adăugarea sau ştergerea de noi proprietăți.

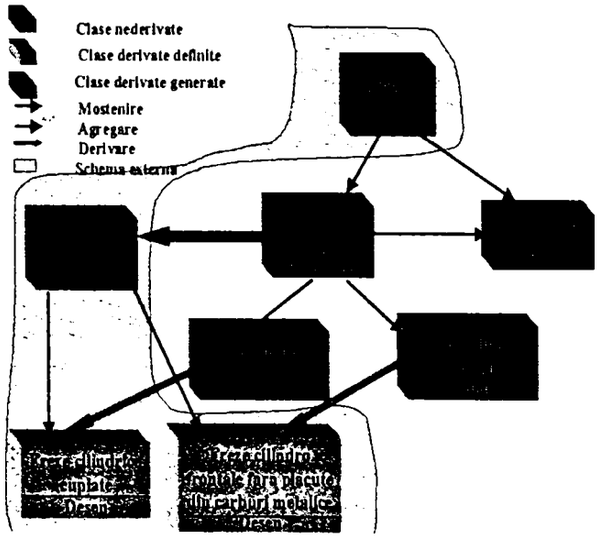


Fig 4.21 Schemă externă

Dacă facem referinţă la exemplul nostru, considerând clasa "Freze" ca şi clasă transformabilă şi restul de clase netransformabile ca în figura 4.20, schema externă corespunzătoare, este prezentată în figura 4.21. Astfel putem vedea că, clasa "Freze" este înlocuită în schema externă, de clasa "Freze1". În acest caz numai clasa necesară pentru schema externă a fost generată.

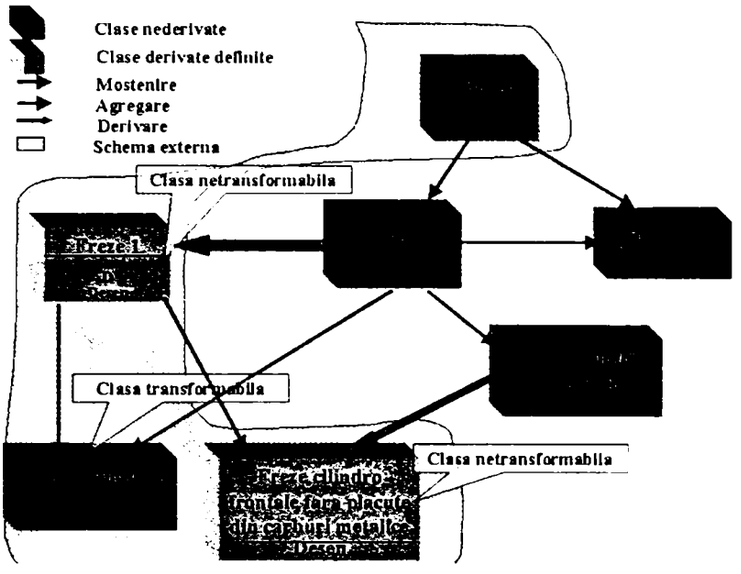


Fig 4.22 Schemă externă

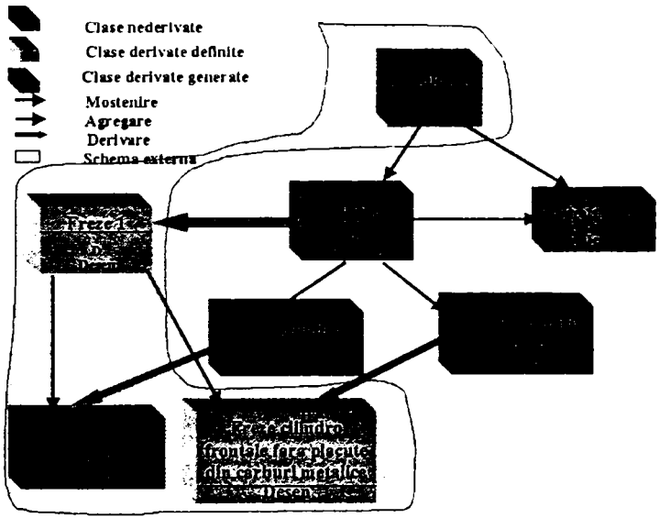


Fig 4.23 Schemă externă

Pot fi şi alte căi pentru definirea schemei externe aşa cum se vede în figura 4.22. În acest caz clasele derivate "Freze1" şi "Freze cilindro frontale fara placute din carburi metalice" sunt definite explicit, ele fiind considerate netransformabile, iar clasa "Freze cilindrice" este considerată transformabilă. În aceste condiţii schema externă generată (fig 4.23) este identică cu cea din figura 4.21. Cu toate acestea mulţimea claselor generate este diferită pentru că, clasa

"Freze cilindrice cuplate" a fost generată din clasa transformabilă "Freze cilindrice"

4.3.3 Semanticile obiect conservat şi obiect generat

O clasă derivată este definită cu semantica *obiect conservat* dacă ea conține numai obiecte extrase din clasele sursă. O clasă este definită cu semantica *obiect generat* dacă ea conține obiecte noi generate în procesul de definire, obiectele noi trebuie să fie identificate prin noi identificatori generați. În [Kim 95], [Monk 94], [Naja 95], clasele derivate sunt definite prin semantica obiect conservat. În [Kifer 92] se consideră că definirea claselor derivate se poate face folosind doar semantica obiect generat. În [Abite 91], [Dayal 89], [Santo 95] se arată că există sisteme care permit folosirea ambelor semantici, pentru definirea claselor derivate.

4.3.4 Identificatori ai obiectelor în clasele derivate

Fiecare obiect derivat sau nederivat este reprezentat prin identificatorul său (OID). Acesta este definit funcție de alt identificator de obiect sau funcție de o valoare pe care o poate lua o anumită proprietate la un moment dat. În cazul în care identificatorul este funcție de un alt identificator există două cazuri funcție de semantica cu care au fost generate obiectele. Definirea claselor derivate prin semanica obiect conservat, poate fi considerat un caz particular, identificatorii obiectelor din clasele derivate, depind de identificatorii obiectelor din clasele de bază. În cazul semanticii obiect generat, conform [Garda 96], [Kifer 92], noi identificatori ai obiectelor din clasele derivate, pot fi definiți ca funcții a identificatorilor obiectelor din clasele de bază. Astfel în [Kifer 92] generarea identificatorilor pentru noile obiecte din clasele derivate se face cu ajutorul unei funcții specifice pentru fiecare clasă derivată. Această funcție trebuie să returneze o singură valoare, pentru fiecare set de parametrii, valoare care trebuie să fie unică în baza de date. O altă posibilitate, este de a defini obiecte noi, funcție de valorile proprietăților sau combinații între valorile proprietăților şi identificatorii altor obiecte [Hull 91], [Santo 94]. Astfel conform [Abite 91], se poate considera că identificatorul unui obiect din clasele derivate, este generat de mulțimea atributelor obiectului

4.3.5 Transmiterea modificărilor obiectelor din clasele derivate

În scopul de a transmite modificările obiectelor din clasele derivate, de la obiectele din clasele de bază, din care sunt derivate clasele ce conțin obiectele, trebuie definită o conexiune între clasele existente. Dacă definirea claselor

derivate este îndeplinită exclusiv cu semantica obiect conservat, această conexiune este imediată şi directă [Heuer 91], [Kim 95] . Dacă clasa derivată este definită prin semantica obiect generat, trebuie menținută legătura între identificatorul obiectului din clasa derivată şi obiectul din clasa care defineşte clasa derivată. Transmiterea modificărilor se poate face în două moduri: automat de către sistem sau cu ajutorul metodelor din clasele derivate. Modul automat de transmitere al modificărilor este cel mai simplu. Totuşi datorită numărului mare de conexiuni care trebuie memorate, în cazul în care se foloseşte semantica obiect generat, este recomandat să nu se folosească . Totuşi în [Guerr 97] este propus mecanismul automat în cazul în care se foloseşte semantica obiect conservat. Există şi posibilitatea folosiri metodelor conform [Ander 93], [Runde 92-4], în special în cazul în care există ambiguități privind modul de transmitere a identificatorilor de obiecte.

Concluzii

Sistemele de baze de date relaționale, prevăd un număr mic de tipuri de date ca de exemplu şir de caractere, intreg, real şi structuri de date conceptuale ca de exemplu articole, relații, care au fost prevăzute în special pentru aplicații de tip economic. Aceste structuri de date sunt necorespunzătoare pentru reprezentarea unor structuri bogate de tipul obiectelor complexe. În plus DBMS-urile convenționale au un număr restrâns de operații pentru manipularea articolelor sau a relațiilor şi un număr redus de metode care pot implementa operațiile existente într-un DBMS. Cerințele unei aplicații obiectuale sunt adesea imposibil de rezolvat de către un DBMS relațional. Aşa cum s-a arătat bazele de date orientate obiect OODBMS împrumută caracteristici de la bazele de date convenționale precum şi de la limbajele de programare orientate obiect. Scopul unei astfel de baze de date este de a simplifica pe cât posibil dezvoltarea unei aplicații şi eliminarea problemei tranzacțiilor.

Studiul materialului prezentat în primele capitole reflectă faptul că dezvoltarea OODBMS-urilor se face pe două direcții importante:

- Dezvoltarea conceptelor actuale referitoare la DBMS-uri;
- Dezvoltarea conceptelor legate de limbajelor de programare şi în special acelea care se referă la persistență şi partajare.

Dezvoltarea conceptelor actuale referitoare la DBMS-uri este legată de cunoaşterea conceptelor fundamentale ale DBMS-urilor relaționale:

- Modele de date: tuple, atribute, relații;
- Tranzacțiile;
- Interogări care se realizează asupra datelor persistente în baza de date prin limbajul SQL.

Trecerea de la relațional la obiectual se face în primul rând prin definirea unor modele semantice de tipul obiect al lumii reale. Astfel obiectul este definit unic, şi identificat prin OID. Apar concepte noi prin care atributele pot fi moştenite.

Cercetările efectuate pentru a dezvolta aceaste concepte s-au făcut în principal pe următoarele direcții:

1. tipuri de sisteme care incorporează mecanisme puternice de moştenire şi în special moştenirea multiplă;
2. limbaje de interogare care cuprind operații puternice asupra obiectelor;
3. tehnici de acces, de clustering şi buffering pentru obiecte de dimensiuni diferite;
4. optimizarea interogărilor;
5. implementarea unor proceduri complexe de protecție a obiectelor;
6. implementarea unor obiecte adecvate de proiectare şi programare

Din punct de vedere al limbajelor de programare apare ca o necesitate importantă folosirea unor noi abstracizări ale datelor şi implementarea tuturor conceptelor obiectuale

Limbajul de definire al obiectelor ODL este similar cu limbajul de definire al datelor(DDL), el are rolul de a definii toate obiectele bazei de date în special în mediile soft orientate obiect care respectă standardul ODMG. El asigura compatibilitatea între diverse medii SGDBOO.

Limbajul de interogare al obiectelor (OQL) este o extensie a limbajului SQL, el putând fi folosit ca un limbaj autonom sau ca un limbaj înglobat într-un mediu soft pentru care este definită o legătură de tip ODMG.

Cele mai importante direcții în care se face cercetarea pentru dezvoltarea conceptelor referitoare la limbajele de programare sunt:

1. Modul în care este memorat obiectul, legătura dintre modul de memorare tranzient şi modul de memorare persistent. Memorarea tranzientă se realizează cu ajutorul unor pointeri, pe când memorarea persistentă se realizează cu ajutorul referințelor la obiect. Legătura între cele două moduri trebuie să se realizeze automat de căte limbaj. Efectul major al acestui concept este performanța pe care trebuie să o asigure tranzacția;
2. Memorarea obiectelor şi asigurarea gestiunii zonelor tampon de memorare a acestora. Aceste tehnici se bazează pe asigurarea unor pagini de memorare de mărime aproximatv egală;
3. Integrarea limbajelor de programare cu modelele de date. Acest lucru permite ca un obiect server să poată lucra cu mai multe limbaje de programare persistente, realizându-se partajarea obiectului între limbaje;
4. Asigurarea unor interogări care să asigure manipularea obiectelor;

5. Partajarea este o altă caracteristică care necesită ca DBMS să conţină rutine care să asigure controlul concurenţei şi refacerea informaţiei;

Modelul relaţional este simplu dar el are ca model de date tabela (relaţia), iar rezultetele finale sunt efectul unor tranzacţii simple. Modelul relaţional este foarte diferit faţă de modelul obiectual. El nu este proiectat pentru a memora obiecte complexe. Totodată acest model permite implementarea greoaie a principalelor concepte obiectuale ca de exemplu moştenirea, polimorfismul, identitatea obiectelor etc.

Bazele de date obiectuale simplifică foarte mult managementul datelor pentru că ele suportă în mod direct modelul obiectual. Studiile au arătat că aplicaţiile care fac referinţă la bazele de date orientate obiect au nevoie de cod program redus cu mai mult de 50% fată de aplicaţiile care apelează alte modele ale bazelor de date. Aceasta conduce la scurtarea timpului de dezvoltare al aplicaţiei, la creşterea calităţii acestora şi a ciclului de menţinere al aplicaţiilor.

Bazele de date obiectuale permit modelarea mai convenabilă a "mediului extern". Pentru "Sistemul de fabricaţie al Sculelor" (SFS), acest lucru va fi evident, fiecare sculă de un anumit tip va fi o clasă de obiecte care va conţine mai multe instanţe funcţie de proprietăţile lor. Relaţiile dintre obiecte vor fi definite ca şi clase, la fel şi vederile (clasele derivate) care vor fi realizate.

Abilitatea de a extrage proprietăţi comune ale claselor şi de a definii superclase de obiecte, care se pot fi partajate în subclase reduce mult redundanţa OODB-ului. Acelaşi lucru poate fi interpretat folosind noţiunile de clasă derivată sau clasă derivabilă. Aceste caracteristici reprezintă unul dintre avantajele majore ale sistemelor obiectuale.

Existenţa superclaselor şi a subclaselor sau a claselor derivate şi a claselor derivabile, face ca schema obiectuală să fie mult mai fezabilă şi totodată prin moştenire să fie mai intuitivă şi mai structurată.

În capitolele anterioare ale tezei s-a făcut o sistematizare a sistemelor orientate obiect, s-au prezentat principalele lor caracteristici, arhitectura şi standardizarea acestora. Totodată s-a pus un accent deosebit pe definirea schemei conceptuale şi a schemei obiectuale prezentându-se teorii privind modul de elaborare a acestora. S-au prezentat şi dezvoltat noţiunile de clasă derivabilă şi clasă derivată. Aplicabilitatea acestor concepte va fi arătată în capitolul 5 şi totodată vor fi folosite în dezvoltarea aplicaţiei pentru SFS.

S-au studiat elementele componente ale unei scheme externe şi clasele derivate din OODB. Pornind de la arhitectura ANSI/SPARC, s-a reliefat faptul că pentru o arhitectură aplicabilă mediilor relaţionale, schema externă care este parte a acestui concept de arhitectură, poate fi aplicată şi sistemelor obiectuale. În schema externă pot fi introduse clasele derivate.

În capitolul anterior s-au arătat principalele metodologii de realizare a schemelor externe şi a claselor derivate. Întegrarea claselor derivate cu alte clase

în schema obiectuală, posibilitatea de a defini clasele derivate cu semantici diferite, generarea de noi obiecte, transmiterea modificărilor între obiecte, sunt probleme asupra cărora se propun alternative în cadrul tezei.
Dacă la apariția lor conceptele orientate obiect pentru bazele de date păreau că vor revoluționa complet bazele de date, încet, încet, s-a dovedit că o serie de concepte proprii mediului relațional pot fi adaptate mediului obiectual.

În final acesta este şi scopul tezei de a aduce contribuții privind modul de realizare a unei baze de date obiectuale, care să conțină elemente relaționale.

Ca o concluzie generală se poate spune că OODB-urile sunt sisteme moderne care asigură o multitudine de concepte care pot fi aplicate foarte bine sistemelor CAD (proiectare asistată pe calculator), CASE (proiectarea de software asistată de către calculator), OIS (sisteme informatice de birou) şi sistemelor multimedia.

5

Realizarea unei metodologii de definire a claselor derivate şi a schemelor obiectuale

Pornind de la noţiunile teoretice de bază tratate şi sistematizate anterior în teză, se va prezenta în continuare o nouă metodologie simplificată, de definire a schemei externe şi a claselor derivate. Acest lucru se va realiza pornind de la arhitectura specifică mediilor relaţionale adaptată mediilor obiectuale.

5.1 Definirea unei metodologii de realizare a claselor derivate

În paragrafele anterioare au fost prezentate metodologii de definire a schemelor obiectuale. În general pentru a putea definii o clasă derivată trebuie urmărite următoarele etape:

- în schema obiectuală, clasele derivate trebuie integrate cu alte clase, folosind una dintre semantici, obiect generat sau obiect conservat (cap4. paragraf 4.2.1);
- Definirea unor obiecte noi în cadrul claselor;
- Transmiterea modificărilor între obiecte, în clasele derivate şi definirea obiectelor în clasele derivate (cap 4. paragraf 4.3).

Iniţial o schemă externă poate conţine doar clase nederivate. Conform definiţiei, *clasele derivate sunt clase care sunt definite din clasele iniţiale, derivate sau nederivate folosind diverse criterii particulare.* Ele sunt definite în scopul de a fi introduse în schema externă sau în schema obiectuală a bazei de date orientate obiect.

5.1.1 Clase de bază şi obiecte de bază

Aşa cum s-a arătat în capitolul precedent, *clasele din care este definită direct clasa derivată, se numesc clase de bază.* Clasele derivate oferă o nouă interfaţă de acces la informaţiile existente în clasa de bază. Clasele derivate împart accesul la datele conţinute în baza de date împreună cu clasele de bază.

În nici un caz ele nu pot conține informații care nu pot fi accesate din clasele de bază. Obiectele conținute în clasa derivată se numesc *obiecte derivate*. Obiectele din clasele de bază care participă la definirea obiectelor derivate se numesc *obiecte de bază*

O clasă derivată poate fi definită ori prin semantica *obiect conservat*, dacă ea conține numai obiecte ale clasei de bază, ori prin semantica *obiect generat* dacă ea conține obiecte noi, generate de către obiectele clasei sale de bază. Definind clase derivate prin semantica obiect generat este posibil să fie realizate structuri complexe ale informației, care în alte condiții nu ar fi posibile, ca de exemplu transformarea valorilor în obiecte sau agregarea obiectelor. Celălat tip de semantică, poate fi folosit în cazul în care clasa derivată definește o nouă interfață pentru obiectele sale.

5.1.2 Legătura dintre clasa de bază şi clasa derivată

Legătura dintre clasa de bază şi clasa derivată se numeşte *relație derivată* după numele clasei, care se obține din clasa de bază. Relația derivată definește cum se obțin clasele derivate din clasele lor de bază. Ea stabileşte corespondența între obiectele de bază şi obiectele derivate. Ea este folosită, pentru a integra clasele derivate în dicționarul de date al bazei de date. Aceast tip de relație este diferit față de relația de moştenire şi relația de agregare care se definesc în paradigma orientat obiect, având o dimensiune ortogonală.

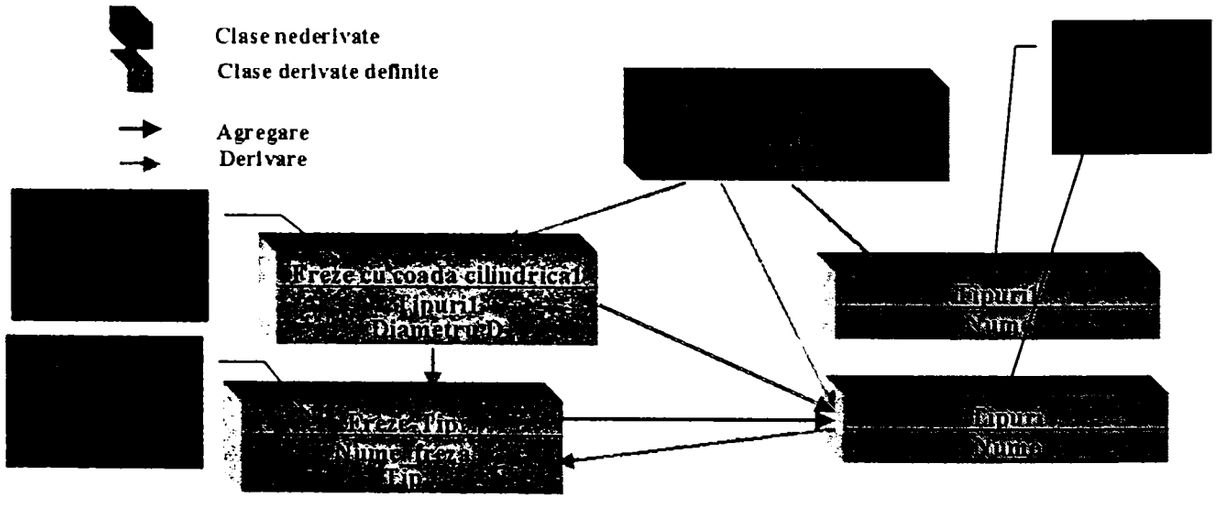


Fig 5.1 Obținere clasei derivate prin semantici diferite

Alte relații asemănătoare sunt: *"derivated from"* definite în [Kim 95] sau *"view derivation"* definite în [Berti 92]. Relația derivată nu apare la aceştia definită în schema conceptuală sau în schema externă. Ea apare doar în dicționarul de date.

În continuare vom prezenta un exemplu, în figura 5.1. Inițial avem clasa de bază, "Freza cu coada cilindrica", care conține proprietatea "Tipuri", ce returnează numele tipurilor de freze cu coada cilindrică, pentru un anumit obiect. Din clasa "Freza cu coada cilindrica" sunt derivate clasele "Tipuri1" şi

"Tipuri" acestea fiind definite prin semantica obiect generat, clasele reprezentând tipuri de freze existente, funcție de anumite criterii.

Clasa "Freza cu coada cilindrica1" este definită prin semantica obiect conservat ea păstrând aceleaşi obiecte ca şi clasa de bază. Această clasă reprezintă aceleaşi concepte ca şi clasa inițială "Freza cu coada cilindrica" dar adaptate la noile cerințe impuse de clasa "Tipuri", cu care este legată printr-o relație de agregare.

Clasa Freze-Tip este definită prin semantica obiect generat şi obiectele sale, sunt diferite perechi, care pot fi definite între obiectele clasei "Freza cu coada cilindrica1" şi tipurile acestora. Acesta reprezintă un nou concept definit pe baza informației existente în două clase derivate. Cu alte cuvinte din două clase derivate s-a obținut o altă clasă derivată, care are obiecte noi diferite de cele inițiale. Folosind acest mod de generare a claselor derivate, dezvoltarea schemei externe devine practic fără restricții, posibilitățile de dezvoltare fiind infinite. Acest exemplu arată următoarele:

- definirea claselor prin semantica obiect conservat în cazul clasei "Freza cu coada cilindrica1", se poate face tot aşa de bine ca şi definirea claselor folosind semantica obiect generat, adică pentru definirea unei clase derivate poate fi folosită oricare din semanticile dorite, rezultatele fiind asemănătoare;
- transformarea valorilor în obiecte în definirea clasei "Tipuri1" şi "Tipuri";
- agregarea obiectelor în scopul de a forma o clasă cu obiecte total diferite din punct de vedere conceptual, în cazul clasei "Freze-Tip".

5.1.3 Identificatorii claselor derivate

Fiecare obiect de bază sau obiect derivat este localizat, prin identificatorul său. Proprietățile obiectului pot defini legătura dintre identificator şi valorile sale sau legătura dintre identificator şi alți identificatori.

Plecând de la exemplul prezentat anterior, în figura 5.2, sunt prezentate elementele clasei nederivate "Freze cu coadă cilindrică" folosind reprezentarea tabelară. Fiecare obiect este reprezentat printr-o linie de tabel, pentru fiecare este definit un identificator (coloana oid) şi proprietățile Nume, Diametru şi Tipuri.

Identificator obiect (OID)	Nume	Tipuri	Diame
F1	**Cilindro frontale**	**N, M, D**	**3**
F2	**Cilindro frontale lungi racordate la colț**	**A/I**	**16**
F3	**Cilindro frontale cap sferic**	**B/I, F/I scurte, F/I lungi**	**12**
F4	**Conice elicoidale cu cap rotunjit**	**C/I scurte, C/I medii, C/I scurte**	**14**
F5	**Conice cap sferic**	**F/I scurte, F/I lungi**	**10**

Fig 5.2 Clasa nederivată "Freze cu coadă cilindrică"

5.1.3.1 Folosirea semantici obiect conservat pentru obținerea clasei derivate

În exemplul din figura 5.1, s-a definit o singură clasă prin semantica *obiect conservat* și anume clasa "Freze cu coadă cilindrică1". În figura 5.3 se observă că fiecare identificator al clasei "Freze cu coadă cilindrică1" este identic cu identificatorul corespunzător din clasa nederivată "Freze cu coadă cilindrică" prezentată în figura 2.
Obiectele definite prin semantica obiect conservat, sunt obiecte care există și în clasa de bază.

Freze cu coadă cilindrică1	
Oid	Obiecte de bază
F1	**F1**
F2	**F2**
F3	**F3**
F4	**F4**
F5	**F5**

Fig 5.3 Identificatorii clasei "Freze cu coadă cilindrică1" obținută prin semantica obiect conservat

5.1.3.2 Folosirea semantici obiect generat pentru obținerea clasei derivate

În cadrul unui sistem pot fi generați identificatori de noi obiecte sau pot fi generate obiecte noi. Putem considera că identificatorul unui obiect derivat este generat de către un set de atribute ale sale. În [Abite 91], aceste atribute se numesc *atribute de memorare*. Aceste atribute, pot să nu facă parte din interfața unei clase derivate, ele pot fi proprietăți interne. În cazul claselor derivate definite prin semantica obiect conservat, putem considera că pentru fiecare obiect al clasei derivate există un atribut intern, care returnează identificatorul obiectului de bază. Din această cauză niciodată nu pot exista într-o clasă derivată două obiecte având valori identice pentru atributele interne. Datorită

acestui fapt obiectele dintr-o clasă derivată pot fi reprezentate prin identificatorul lor sau prin atributul intern.

O clasă este identificată prin valori, dacă obiectele sale pot fi identificate, folosind un set de atribute valori (nu identificatori de obiecte). În acelaşi sens putem defini *o clasă identificabilă prin atribut dacă obiectele sale pot fi identificate folosind o mulțime de atribute. Clasele derivate sunt identificabile prin atribut, deoarece obiectele sale sunt identificate de către atributele interne.* Folosind aceste definiții, reprezentarea claselor arătate în figura 1 este următoarea: pentru clasa "Freze cu coadă cilindrică", clasa "Tipuri" este definită în aşa manieră încât, fiecare obiect este generat funcție de valoarile diferite, dintre cele returnate de proprietățile clasei "Freze cu coada cilindrică", permițînd, transformarea valorilor în obiecte. Rezultatul acestei interpretări este arătat în figura 5.4.

Tipuri			
OID	Atribut de memorare	Obiecte de bază	Nume
T11	**F/I scurte**	**{F3,F5}**	**Forma-F-TipI-S**
T12	**F/I lungi**	**{F3,F5}**	**Forma-F-TipI-L**
T13	**N**	**{F1}**	**Tip-N**
T14	**M**	**{F1}**	**Tip-M**
T15	**D**	**{F1}**	**Tip-D**
T16	**A/I**	**{F2}**	**Forma-A-Tip-I**
T17	**B/I**	**{F3}**	**Forma-B-Tip-I**
T18	**C/I scurte**	**{F4}**	**Forma-C-Tip-I-S**
T19	**C/I medii**	**{F4}**	**Forma-C-Tip-I-M**
T20	**C/I Lungi**	**{F4}**	**Forma-C-Tip-I-L**

Fig 5.4 Clasa tipuri obținută prin semantica obiect generat

Dacă două obiecte ale clasei "Freze cu coada cilindrica" au aceeaşi valoare returnată de proprietățile clasei "Tipuri", ambele sunt obiecte de bază cărora le corespunde câte un obiect derivat. În general toate obiectele de bază pentru care se obțin valori identice pentru atributele interne sunt incluse în mulțimea obiectelor de bază, pentru obiectele derivate, care vor fi generate. În figura 5.5 este arătată reprezentarea obținută pentru clasa "Tipuri1". Identificatorul obiectului de bază şi numele tipului au fost definite ca atribut intern. Pe această cale un obiect nou este generat pentru fiecare valoare returnată de mulțimea proprietăților tipuri, câte unul pentru fiecare obiect al clasei "Freze cu coada cilindrica".

Tipuri1			
OID	Atribut de memorare	Obiecte de bază	Nume
K11	**F3,F/I scurte**	**{F3,F5}**	**Forma-F-TipI-S**
K12	**F3,F/I lungi**	**{F3,F5}**	**Forma-F-TipI-L**

K13	**F5,F/I scurte**	**{F3,F5}**	**Forma-F-TipI-S**
K14	**F3,F/I lungi**	**{F3,F5}**	**Forma-F-TipI-L**
K15	**F1,N**	**{F1}**	**Tip-N**
K16	**F1,M**	**{F1}**	**Tip-M**
K17	**F1,D**	**{F1}**	**Tip-D**
K18	**F2,A/I**	**{F2}**	**Forma-A-Tip-I**
K19	**F3,B/I**	**{F3}**	**Forma-B-Tip-I**
K20	**F4,C/I scurte**	**{F4}**	**Forma-C-Tip-I-S**
K21	**F4,C/I medii**	**{F4}**	**Forma-C-Tip-I-M**
K22	**F4,C/I Lungi**	**{F4}**	**Forma-C-Tip-I-L**

Fig 5.5 Clasa "Tipuri1" obținută prin semantica obiect generat

Clasa "Freze cu coada cilindrica1" reprezentată în figura 5.6 a fost definită prin semantica obiect conservat, un singur atribut intern a fost folosit pentru identificarea obiectului de bază.

Freze cu coada cilindrica1			
OID	**Atribute de memorare**	**Obiecte de bază**	**Tipuri**
F1	**F1**	**F1**	**T13,T14,T15**
F2	**F2**	**F2**	**T16**
F3	**F3**	**F3**	**T11,T12,T17**
F4	**F4**	**F4**	**T18,T19,T20**
F5	**F5**	**F5**	**T11,T12**

Fig 5.6 Clasa "Freze cu coadă cilindrică1" obținută prin semantica obiect conservat

Clasa "Freze-Tip", fig 5.7, a fost definită prin semantica obiect generat, folosind asocierea obiectelor. Tipurile Forma-F-TipI-S şi Forma-F-TipI-L au fost definite între două obiecte diferite ale clasei "Freze cu coadă cilindrică1" care au aceleaşi tipuri.

Freze - Tip				
OID	Atribute de memorare	Obiecte de bază	Tipuri	Freze
M1	**T11, {F3,F5}**	**T11, {F3,F5}**	**T11**	**F3,F5**

Fig 5.7 Clasa "Freze-Tip"

5.1.3.3 Clase care conțin obiecte generate

În general, într-un sistem orientat obiect, legătura între clase nederivate şi clase derivate sau nederivate se face prin:

- moştenire, definind o clasă derivată ca şi subclasă a unei clase derivate existente;
- derivare, definind noua clasă, folosind semantica obiect conservat dintr-o altă clasă derivată.

Folosind cea de a doua variantă, pentru generarea de noi identificatori, poate fi folosit *factorul Skolem* [Hull 91]. În general un factor sau funcționalitatea Skolem distinct, este folosit pentru fiecare clasă derivată nouă. Dând valori la o mulțime de atribute interne factorul generează noi identificatoare de obiecte. Identificatorii generați sunt funcții ale atributelor interne ale clasei derivate. Ne propunem să definim identificatorii obiectelor noi ca şi funcții ale atributelor interne. De aceea două clase diferite, definite prin semantica obiect generat pot avea obiecte în comun fără a fi legate prin relația de moştenire, având acelaşi set de atribute interne. Atributele sunt identificate în mod unic în dicționarul de date independent de numele lor. În general se consideră că proprietățile definite în dicționarul de date au un nume unic. Pentru a realiza acest lucru, intern, este asignat câte un identificator pentru fiecare proprietate. Astfel generarea de identificatori pentru obiectele unei clase depinde doar de atributele interne selectate.

De exemplu considerăm clasa derivată "Tipuri2" care este legată de frezele cu diametrul mai mare sau egal decât 10. Clasa este definită pe baza semanticii obiect generat (clasa de bază este clasa "Freze cu coadă cilindrică"). Ea conține un subset de obiecte ale clasei "Tipuri" (figura 5.4) independent de clasele definite anterior "Tipuri1" sau "Tipuri" (figura 5.8)

Tipuri2			
OID	Atribut de memorare	Obiecte de bază	Nume
T11	**F/I scurte**	**{F3,F5}**	**Forma-F-TipI-S**
T12	**F/I lungi**	**{F3,F5}**	**Forma-F-TipI-L**
T16	**A/I**	**{F2}**	**Forma-A-Tip-I**
T17	**B/I**	**{F3}**	**Forma-B-Tip-I**
T18	**C/I scurte**	**{F4}**	**Forma-C-Tip-I-S**
T19	**C/I medii**	**{F4}**	**Forma-C-Tip-I-M**
T20	**C/I Lungi**	**{F4}**	**Forma-C-Tip-I-L**

Fig. 5.8 Clasa Tipuri2

5.1.4 Definirea obiectelor în clase derivate

Definirea unei clase derivate se poate face printr-o comanda specifică unui anumit mediu de programare orientat obiect. De exemplu pentru mediul Prolog această comandă este derivedClass. Crearea de noi obiecte este controlată de către comanda newobject. În alte medii de baze de date clasele derivate pot fi create cu comenzi specifice (define class, class{ ...}) etc., ele fiind considerate din punct de vedere conceptual clase derivate. Obiectele clasei sunt create cu comenzi specifice mediului de programare obiectuală (add object, create object, new, newobject etc.). Definirea moştenirii sau a derivării prin

moştenire poate fi realizată folosind diverse cuvinte rezervate în contextul limbajului (extends, as, from etc).

Obiectele de bază şi obiectele derivate sunt toate obiectele pe care le definim. Mulțimea obiectelor care participă la definirea şi identificarea unui obiect derivat este un subset al mulțimii obiectelor de bază. Relația de derivare stabileşte corespondența între obiectele de bază şi obiectele clasei derivate. Unele dintre obiectele de bază, determină identitatea obiectelor derivate. De asemenea identificatorii obiectelor sunt returnați de către proprietăți care formează atribute interne, obținute dintr-o submulțime a obiectelor de bază. Unele obiecte de bază sunt folosite în scopul de a definii proprietăți ale obiectelor derivate. Din această cauză există două înțelesuri pentru relația derivată:

- derivarea prin identitate, dacă obiectele clasei de bază participă la definirea identității obiectelor din clasa derivată;
- derivarea prin valoare dacă obiectele clasei de bază nu participă la definirea identității obiectelor derivate.

5.2 Clase nederivabile sau derivabile parțial

În urma analizei făcute, asupra modului în care sunt create clasele derivate pot apare, întrebări legate de modul în care informațiile existente în clasele derivabile sau nederivabile pot fi la rândul lor derivabile sau nederivabile.

5.2.1 Informații nederivabile în clase

Clasele de bază ale unei clase derivate pot fi la rândul lor alte clase derivate. Ele pot fi clase derivabile sau nederivabile. Informația conținută în clasele nederivabile este obținută direct din baza de date. Prin aplicarea tranzitivității relației derivate informația despre clasele derivate este calculată pe baza datelor memorate în baza de date, clasele derivate oferind o nouă interfață pentru aceste date. Cu alte cuvinte clasele derivate folosesc în comun datele din baza de date, împreună cu clasele de bază. În nici un caz nu pot fi derivate clase care nu conțin informații din clasele de bază.

Uneori utilizatorii finali cer schimbarea informațiilor. Aceştia au nevoie de informație nouă care nu poate fi derivată din informația existentă în baza de date. Această informație nouă trebuie incorporată în baza de date. Adăugarea de informație se poate face, fie definind clase noi nederivabile, fie modificând clasele existente, astfel încât informația să poată fi inclusă. Modificarea clasei existente poate fi deranjantă în unele cazuri, dacă noile clase sunt derivate sau dacă există programe care le folosesc.

5.2.2 Clase derivate parțial

Pentru a spori capacitățile de transformare a claselor în schemele externe, o soluție ar fi aceea de a permite claselor derivate, să conțină informații nederivabile, adică să existe clase parțial derivabile. Informația nederivabilă este adăugată la informația obținută din clasele de bază, astfel se evită nevoia de a modifica definirea altor clase. Definirea claselor parțial derivate ne permite să definim clase derivate cu capacități de transformare sporite, din clasele de bază, fără ca restul de clase să fie afectate. Informația conținută în clasele de bază, este conținută în partea nederivabilă a claselor derivabile parțial.

5.2.2.1 Elemente nederivabile în intensia clasei

Unele sisteme permit definirea elementelor nederivate în intensia unei clase parțial derivate [Ra 95], [Guerr 97], adică putem avea clase parțial derivate cu proprietăți nederivate. Astfel de proprietăți pot lua anumite valori inițiale. O posibilitate suplimentară necesară, este aceea de a modifica tipul proprietății în sensul creşterii capacității de informare, adică o proprietate simplă devine multivaloare în clasa parțial derivată. O proprietate definită ca întreg poate deveni reală în noua clasă. Valoarea inițială este aceea obținută din obiectul de bază. Informația nederivată definită în intensia clasei, trebuie memorată undeva pentru fiecare nouă apariție în noua clasă definită.

5.2.2.2 Elemente nederivabile în extensie

O clasă derivată este definită prin semantica obiect conservat, care ascunde eventualele constrângeri, definite în strânsă legătură cu clasa de bază. Din această cauză, o clasă derivată poate conține obiecte care nu au fost incluse în clasa de bază. Considerăm exemplul din figura 9 unde clasa "TIPURI" a fost definită prin semantica obiect generat din proprietatea TIPURI a clasei "FREZE CILINDRO FRONTALE". Prin această transformare, înțelesul valorii tipuri a fost schimbat. Astfel din valoare ea a fost schimbată într-o proprietate multivaloare a unui obiect. Având conceptul tipuri reprezentat ca valoare, putem spera ca acesta să apară doar ca o instanță a clasei "FREZE CILINDRO FRONTALE". O dată ce clasa TIPURI a fost definită şi există restricția privind dependența dintre aparițiile în clasa TIPURI şi aparițiile în noua clasă "FREZE CILINDRO FRONTALE1", nu sunt probleme. Dacă însă restricțiile nu sunt definite sau dacă schema externă definită, conține clasa TIPURI, dar nu conține clasa "FREZE CILINDRO FRONTALE1", este posibilă inserarea de obiecte noi în clasa TIPURI, fără ca acestea să aibă asociată nici o instanță în clasa "FREZE

CILINDRO FRONTALE1". Problema este că o operație oarecare nu este permisă de către schema de bază.

5.2.2.3 Extensia locală pentru o clasă parțial derivată.

O clasă a fost creată în scopul de a mări capacitatea de informare respectând clasele de bază. Astfel clasele de bază, nu prevăd suport pentru toate informațiile pe care le conțin noile clase. O cale pentru rezolvarea acestei probleme, este aceea de a permite definirea unor clase parțial derivate cu extensii locale nederivate. Bazat pe termenul definit de [Brats 92], extensia locală a unei clase parțîal derivate, conține elemente nederivate care sunt definite atât în extensia clasei cât şi în intensia acesteia.
Un exemplu concret este următorul. Considerăm clasa TIPURI. Un obiect al clasei, adică un tip, este asociat cu o apariție în clasa "FREZE CILINDRO FRONTALE1". Considerăm clasa TIPURI ca o clasă parțial derivată. Dacă introducem un nou tip cu valoarea, "Freze cilindro frontale cu coadă cilindrică (Weldon)", un obiect cu acest nume, va fi creat în extensia locală a clasei "TIPURI". Acest obiect a fost creat dar dacă în clasa de bază nu există nici o valoare cu cea introdusă, el nu este asociat cu nici un obiect de bază.
Este necesar ca un utilizator să modifică obiectele în clasa "FREZE CILINDRO FRONTALE" astfel încât nici un obiect, să nu rămână fără un tip asociat. Dacă clasa "TIPURI" este manipulată independent, acest lucru poate fi un inconvenient. În procesul de transmitere al modificărilor efectuate obiectul generat în clasa TIPURI nu este şters ci mai degrabă este transformat într-un obiect nederivat. În acelaşi mod un obiect derivat poate deveni nederivat sau un obiect nederivat poate deveni derivat. Dacă în clasa "FREZE CILINDRO FRONTALE", tipul "Freze cilindro frontale cu coadă cilindrică (Weldon)", este definit ca o valoare a proprietați tipuri a unui obiect, nu va putea fi generat un nou obiect în clasa TIPURI, dar în schimb un obiect nederivat poate deveni derivat şi asociat cu un obiect de bază. Extensia unei clase parțial definită este transmisă extensiei derivate obținută din clasele de bază şi din extensia locală. Obiectul extensiei locale poate face parte dintr-o extensie derivată, la fel cum obiectele extensiei derivate pot face parte dintr-o extensie locală.

5.2.2.4 Gradul de propagare ale legăturilor

Evoluția claselor adică intensia şi extensia sunt considerate separat. Dacă consider numai extensia, aceasta înseamnă că dacă inserăm un obiect nou în clasa de origine, trebuie să includem un obiect nou şi în clasa de destinație. Acest lucru este valabil în cazul în care există o legătură între clasa de origine şi clasa destinație. Clasa destinație conține toate obiectele clasei origine, iar pe lângă acestea poate conține şi alte obiecte neexistente în clasa de origine.

Obiectele din cele două clase pot avea proprietăți comune, dar și proprietăți diferite. Dacă considerăm extensia claselor, există mai multe posibilități:

- Clasa destinație poate conține toate extensiile clasei origine, dar nu și toate proprietățile și metodele, intensia clasei;
- Ambele clase originea și destinația pot avea elemente comune dar nici una nu conține instanțele celeilalte clase. Acest lucru poate semnifica că instanțele clasei origine pot deveni instanțele clasei destinație în anumite condiții precis definite. Astfel se poate definii fluxul de informație între clasele de bază și clasele derivate și invers.

5.2.3 Concluzii privind realizarea metodologiei de definire a claselor derivate

Definirea claselor derivate prin semantica obiect conservat sau obiect generat face posibilă definirea unei noi interfețe pentru obiectele existente sau permite reorganizarea datelor în structuri complexe. Totodată permite transformarea valorilor în obiecte sau agregarea obiectelor. În scopul de a permite aceste concepte privind transformarea, identificatorii noilor obiecte au fost generați din alte obiecte prin valorile atributelor. Aceste lucruri absolut noi au fost dezvoltate în acest capitol. Relația de derivare este definită între clase derivate și mulțimea claselor de bază. Ea este folosită pentru a integra clasele derivate în dicționarul de date, al bazei de date obiectuale. Legătura dintre obiectele derivate și obiectele clasei de bază este definită folosind relații derivate în identificare și în valoare.

5.3 Definirea unei metodologii de realizare a schemei externe

Schemele externe sunt derivate din schema conceptuală a bazelor de date. Ele pot fi folosite pentru a simula schimbările în schema conceptuală a bazei de date. Ea poate conține clasele conținute în schema conceptuală, precum și clase derivate direct sau indirect din clasele de bază ale schemei conceptuale. Clasele derivate pot fi definite prin semantica obiect conservat sau obiect generat. De multe ori informațiile care ajung la utilizatori se schimbă. Ei au nevoie de informații care nu mai pot fi derivate din cele existente în baza de date. O soluție pe care o propunem este definirea claselor parțial derivate.

5.3.1 Informația în schema externă

Schemele externe, provin din schema conceptuală. Fiecare schemă externă descrie o parte din informația existentă în baza de date, în scopul de a satisface necesitățile utilizatorului referitor la informație. Schema externă poate include clase definite în schema conceptuală sau clase derivate. Informația reprezentată în clasele derivate se regăsește în schema conceptuală. *Cu toate că*

în literatura de specialitate [Tresc 92], este negat faptul că o schemă externă poate să includă clase derivate prin semantica obiect generat, consider acest lucru adevărat. De fapt acest lucru a fost arătat anterior, când s-a arăta o metodologie nouă de obținere a claselor derivate. Din punct de vedere al schemei externe, vom exemplifica acest lucru considerând schema externă din figura 5.9.

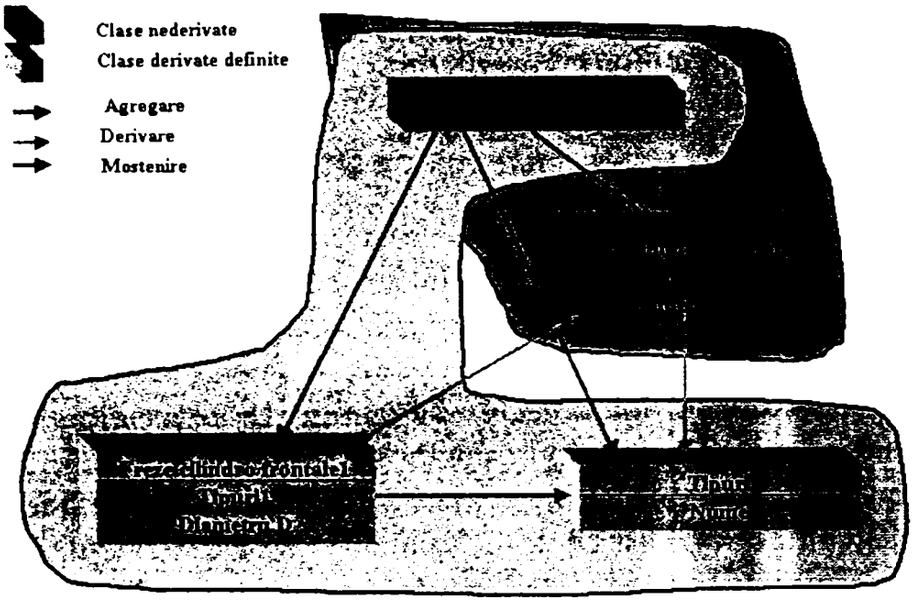


Fig 5.9 Scheme externe

Clasa nederivată "FREZE CILINDRO FRONTALE" are proprietatea "tipuri" care returnează numele tipului de freză cilindică pentru fiecare obiect. Clasa TIPURI derivată din clasa "FREZE CILINDRO FRONTALE" este definită prin semantica obiect generat. Această clasă reprezintă tipurile de freze cilindrice existente pentru fiecare obiect. Clasa "FREZE CILINDRO FRONTALE1" a fost definită prin semantica obiect conservat din clasa "FREZE CILINDRO FRONTALE". Această clasă reprezintă aceleaşi concepte ca şi clasa inițială, dar adaptate la noua reprezentare cerută de clasa "TIPURI", cu care este legată printr-o relație de agregare.

Luând în considerare schema formată din clasa "FREZE CILINDRO FRONTALE" şi noua schemă externă compusă din clasele "FREZE CILINDRO FRONTALE1" şi "TIPURI", dacă este definită o restricție între clasele "FREZE CILINDRO FRONTALE1" şi "TIPURI", în sensul că nici un obiect nu poate exista în clasa "TIPURI" fără să fie legat prin relația de agregre cu un obiect din clasa "FREZE CILINDRO FRONTALE1", puterea de informare din ambele scheme este identică. În acord cu definirea dominației schemelor [Abite 88], schema externă fiind definită din schema originală, toate instanțele sale pot fi obținute din schema originală şi toate informațiile care pot fi conținute în schema originală pot fi obținute şi din noua schemă externă.

Dacă nici o restricție nu este definită între clasa "FREZE CILINDRO FRONTALE1" şi "TIPURI", clasa "TIPURI" poate conține obiecte, care nu sunt în legătură cu valorile proprietății tipuri, din clasa "FREZE CILINDRO FRONTALE". Din această cauză această schemă va domina schema originală.

Ca şi un caz particular al unei scheme externe, poate fi considerată o schemă care să conțină doar clasa "TIPURI". Pentru acest caz clasa este definită prin semantica obiect generat, fiecare obiect al său fiind legat cu valorile atributelor obiectelor din clasa de bază. Dacă un nou atribut este adăugat la clasa "TIPURI", atunci o nouă valoare trebuie adăugată atributului corespunzător unui obiect din clasa "FREZE CILINDRO FRONTALE". Prin relații de derivare acest lucru poate fi posibil. Posibilitățile de modelare a clasei sunt date de partea ei structurală, adică de setul de atribute şi obiecte precum şi de comportarea sa, adică setul de metode care pot fi aplicate obiectelor. Cu alte cuvinte putem avea o schemă externă care să conțină o singură clasă, dar această clasă trebuie să îndeplinească o serie de condiții, în cazul în care este o clasă derivată.

5.3.2 Mediul în care evoluează schema externă

Prin definiție informația conținută în schema externă este obținută din schema conceptuală. Schema conceptuală include toate clasele nederivate definite în dicționarul de date. De asemenea schema conceptuală poate conține şi clase derivate. În cazul în care informația nederivată, sub formă de clase noi nederivate sau clase parțial derivate, este cerută de către utilizatorii finali, informația nou cerută, poate fi inclusă în schema externă, fără a fi inițial prevăzută în schema conceptuală.

5.3.3 Mediul de testare al schemei externe

În procesul de proiectare al schemei, pot apare în mod continu noi clase. Aceasta înseamnă că, schema conceptuală este modificată continu, până se ajunge la versiunea finală. Datorită acestor cauze mediul de testare al schemei este folosit foarte des. Astfel în etapele proiectării sunt definite *scheme temporale* care includ informații nederivate, fără a afecta schema conceptuală. Când un utilizator acceptă o schemă temporală aceasta devine schema finală. De aceea schema conceptuală, este afectată numai de definirea claselor parțial derivate sau de noile clase nederivate ce vor fi incluse în schema externă.

5.4 Evoluția schemei conceptuale de la schema temporală la schema finală

Pentru a transforma o schemă temporală într-o schemă finală trebuie parcurse următoarele etape:

- schema conceptuală este modificată în scopul de a conține informație nederivată;
- schemă externă temporală va deveni o schemă externă derivată din schema conceptuală, adică informațiile nederivate devin derivate;
- schemele externe inițiale nu vor fi afectate de către transformarea schemei conceptuale.

De exemplu în figura 9 numai clasele derivate "FREZE CILINDRO FRONTALE"1 şi TIPURI vor fi incluse în schema conceptuală pentru că ele domină schema compusă exclusiv de clasa "FREZE CILINDRO FRONTALE". Clasa "FREZE CILINDRO FRONTALE" va deveni clasă derivată.

În alte cazuri clasele ce vor fi adăugate la schema conceptuală nu sunt clase care apar în schemele temporale. Aceste clase sunt definite din clasele existente la fel cum sunt definite şi clasele din schemele temporale.

5.5 Definirea unor algoritmi pentru generarea schemelor externe

În continuare ne propunem să definim doi algoritmi pentru generarea schemelor externe funcție de modul în care sunt considerate clasele: transformabile şi netransformabile. Pentru definirea algoritmului, se ține cont de cele prezentate în cap.4 paragraf 4.2, 4.3.

5.5.1 Algoritm pentru schema externă cu toate clasele netransformabile

Acest algoritm se referă la setul de clase considerate netransformabile şi care formează clasele de bază. Schema obiectuală trebuie să fie validă şi să prezinte proprietatea de închidere (closure) în moştenire. Paşii algoritmului sunt următorii:

- *Definirea setului inițial de clase în dicționarul de date*;
- *Definirea setului de clase care va fi introdus în schema externă*;
- *Definirea unei noi mulțimi de clase*, care trebuie să conțină clasele corespunzătoare din cele două seturi de mulțimi. Noul set de clase conține clasele din setul inițial, precum şi noile clase care au fost adăugate. Clasele care vor fi adăugate trebuie să fie legate prin relații cu clasele din setul inițial. Setul de relații dintre clasele inițiale şi restul claselor se găseşte în dicționarul de date. Rezultatul care se obține după acest pas poate fi un rezultat temporar, folosit în paşii următori ai algoritmului.
- Fiind dat setul inițial de clase, se poate spune că noul set de clase, formează o schemă obiect minimală, obținută din setul inițial de clase. Noul set de clase este închis prin relația de moştenire, adică fiecare pereche de clase din setul inițial şi setul adăugat conțin proprietăți comune. Dacă o clasă care aparține noului set de clase, nu conține

proprietăți comune din setul inițial şi cel adăugat, ea poate fi generată prin derivare. Cu alte cuvinte trebuie realizată comparea claselor din setul inițial, cu toate clasele care vor fi adăugate, pentru a fi realizat noul set de clase care va aparține schemei externe. Fiind date clasa1 şi clasa2 care aparțin la familia de clase FamClas1 pot apare următoarele cazuri:

- Dacă $clasa1 \cap clasa2 = clasa1$ atunci clasa2 este subclasă a lui clasa1, legătura dintre clase este de tipul *is_a*;
- În cazul în care $clasa1 \cap clasa2 = clasa2$, clasa1 este subclasă a lui clasa2 şi legătura dintre clase de tipul *is_a;*
- Dacă $clasa1 \cap clasa2 = clasa3$ unde clasa3, este diferită de clasele clasa1 şi clasa2, atunci:
  - Dacă clasa3 este inclusă în setul inițial, atunci clasa3 trebuie să aibă relație de moştenire şi cu clasa1 şi cu clasa2;
  - Dacă clasa3 este o clasă care nu a fost definită în setul inițial, dar a fost derivată o clasă cu aceeaşi intensie (cap. 4 paragraful 4.2.3.3.2), atunci clasa3 se modifică, pentru a satisface extensia (cap. 4 paragraful 4.2.3.3.2) şi sunt adăugate relațiile de moştenire necesare. Cu alte cuvinte sunt generate clase cu aceleaşi proprietăți;

În scopul de a obține o schemă externă în care clasele respectă relația de moştenire, trebuie analizate toate perechile de clase din setul inițial şi setul de clase care va fi adăugat. Dacă între două clase există relația de moştenire, atunci legătura dintre clase este inclusă în setul de legături existent. Printre aceste legături, pot exista legături redundante care trebuie eliminate.

5.5.2 Algoritm pentru realizarea schemei externe folosind clase transfomabile

Acest algoritm oferă posibilitatea de a declara o clasă transformabilă sau netransformabilă (cap.4 paragraf 4.2.3.1) în scopul de a construi schema externă. Clasa netransfomabilă a fost adăugată direct la schema externă. Clasele transformabile pot fi înlocuite prin alte clase existene sau într-un mod nou, printr-o clasă derivabilă. Mecanismul de definire se bazează pe faptul că un număr de clase sunt declarate netransfomabile şi formează schema externă de bază iar restul de clase pot fi transformabile.

Algoritmul conține următorii paşi:

1. *Obținerea setului inițial de clase transformabile şi netransformabile.* Se selectează o mulțime de clase din dicționarul de clase care se consideră transformabile şi netransformabile. Acestea formează

schema inițială. Din setul inițial sunt selectate separat, clasele transformabile şi cele netransfomabile;

2. *Definirea proprietății de închidere a unei relații* pentru schema considerată. Fiecare proprietate a unei clase depinde de alte proprietăți ale claselor sau de alte proprietăți ale aceleiaşi clase. Proprietatea poate fi un atribut sau o metodă a clasei. Este necesar ca în schema externă să apară toate proprietățile claselor folosite direct sau indirect de către clasele care formează schema externă. Trebuie obținută proprietatea de închidere (cap. 4 paragraful 4.2.3.4). Dacă unele proprietăți sunt referite de către clase care nu fac parte din schema externă, atunci există următoarele posibilități:
   - Clasele respective vor fi incluse în schema externă;
   - Clasele respective vor fi redefinite, adică vor fi derivate, în scopul de a avea doar acele proprietăți, de care este nevoie;
   - Clasele care au fost deja incluse în schema externă, vor fi şi ele redefinite pentru a face legătura cu noile clase derivate.

   Practic acest pas se realizează astfel: se determină setul de clase netransformabile. Clasele transformabile referite de către clasele netransformabile, devin netransformabile. Clasele transformabile selectate inițial, care sunt referite de către alte clase transformabile, devin netransformabile. Această schimbare din transformabil în netransformabil se realizează prin eliminarea tuturor referințelor externe.
3. *Obținerea proprietății de închidere pentru relația de moştenire (cap4. 4.3.1.2.4)* impune următoarele condiții: schema care se obține este formată exclusiv din clase derivate, integrarea claselor transformabile, unificarea claselor transformabile
4. *Integrarea claselor transformabile.* Componenta determinantă a unei clase transformabile este extensia. De aceea două clase transformabile cu aceeaşi extensie pot fi aceeaşi clasă. Grupuri de clase transformabile pot fi create prin subsumarea extensiilor (cap 4. paragraful 4.2.3.3.2). Fiecare grup este reprezentat printr-o clasă care are extensia claselor care formează grupul, iar intensia clasei este formată prin uniunea intensiilor claselor din grup. În acest fel integrând clasa reprezentativă a grupului, sunt integrate toate clasele grupului. Clasele transformabile integrate în schemă, sunt considerate ulterior, netransformabile, în procesul de integrare al claselor rămase.
   De exemplu considerăm clasa1, clasă transformabilă care va fi integrată. Dacă clasa transformabilă este subsumată în extensie de altă clasă, clasa2 netransformabilă şi care aparține schemei, atunci clasa1

moşteneşte proprietăţile lui clasa2 şi mai mult decât atât îşi menţine propriile proprietăţi. Oricum dacă clasa1 subsumează în extensie o clasă clasa2 din schemă, clasa2 netransformabilă condiţionează structura clasei transformabile clasa1, în sensul că orice proprietate a acesteia, care nu este definită în clasa2, trebuie eliminată. Când o clasă transformabilă este integrată în schemă ea devine netransformabilă. Cu alte cuvinte o clasă transformabilă poate prin integrare, să determine pierderea unor proprietăţi, a unei clase netransformabile. În primul rând este necesară integrarea claselor transformabile care nu sunt subsumate în extensie, de către alte clase transformabile. În acest caz pot apare două cazuri de transformare:

1. *Clasă transformabilă care conţine subclase în schemă.* În acest caz clasa transformabilă clasa1 ce va fi integrată, subsumează în extensie toate clasele clasa2, care fac parte din schemă. Clasa1 va fi superclasă pentru clasa2 şi subclasă pentru toate superclasele clasei, clasa1. Intensia clasei1, va fi condiţionată de către intensiile superclaselor obţinute din intensia clasei2;
2. *Clasă transformabilă fără subclase*, în schemă. În acest caz se adaugă la proprietăţile clasei clasa1, proprietăţile claselor care o subsumează în extensie

Cei doi algoritmi prezintă următoarele caracteristici:

- Există două faze ale integrării:
  - clasele derivate sunt integrate direct în dicţionarul de date prin relaţia de derivare;
  - apoi clasele selectate, sunt integrate în schema externă folosind relaţiile de moştenire;
- noul concept de clasă transformabilă sau clasă netransformabilă simplifică procesul de definire a schemei externe pentru că permite generarea directă a claselor de care este nevoie în schema externă, evitând definiţii complexe;
- nu este necesară definirea unor subscheme ale schemei conceptuale;

Implementarea bazei de date pentru sistemul de fabricaţie al sculelor va ţine cont de aceşti algoritmi. Implementarea comparativă se va face într-un sistem pur obiectual (Jasmine CA) şi un sistem relaţional obiectual Visual Fox

5.6 Definirea schemelor externe în standardele ODMG

În noul standard ODMG 3.0, apărut în ianuarie 2000, nu apare definirea schemei externe. Standardul permite definirea interogărilor similar cu mecanismul de definire a acestora din RDBMS-uri. Dezvoltarea unui mecanism pentru definirea schemelor externe într-un OODBMS impune şi realizarea unui mecanism pentru definirea claselor derivate, pentru integrarea claselor derivate,

de selectare a unei sintaxe pentru definirea schemelor externe şi dezvoltarea unor algoritmi pentru generarea schemelor externe. Toate aceste fac să ne gândim, că se poate realiza schema externă păstrând caracteristicile RDBMS-urilor.

Aşa cum am arătat anterior în OODBMS-uri schema externă are o structură ierarhică, similară cu schema conceptuală. Aceasta permite integrarea claselor derivate în ierarhia de clase, problemă cunoscută sub numele de *"Problema poziţionării claselor derivate"*, ea putând fi făcută fie manual, fie automat. Această problemă constă, în identificarea claselor derivate, în contextul ierarhiei de clase în care intervin şi clasele de bază. Relaţiile între clase sunt relaţii de agregare sau relaţii de tipul "is-a". Faptul că există relaţii de tip clasă subclasă impune existenţa subtipurilor, precum şi existenţa unui subset sau superset de relaţii între mulţimea instanţelor claselor derivate şi a claselor de bază. Problema poziţionării apare în momentul în care clasa derivată, nu este nici sublasă, nici superclasă a unei alte clase de bază. Acest lucru generează ambiguităţi şi impune generarea unei clase intermediare [Runde 92-1] sau definirea unor noi relaţii între clasele derivate [Berti 92] ca de exemplu relaţia derivată sau relaţia "may-be".

Referitor la mecanismele de generare a schemelor externe şi a claselor derivate, există două concepte fundamentale:

- Schemele externe sunt subscheme ale schemei conceptuale [Runde 92-1], [Heuer 91-1], adică clasele derivate trebuie incluse în schema conceptuală, folosind paradigma obiectuală. Pentru a genera clase derivate care nu sunt subclase sau superclase a unei clase de bază, este necesară generarea unei clase intermediare, care este introdusă la rândul ei în schema conceptuală, crescând complexitatea acesteia. Avantajul acestui concept constă în faptul că în schema conceptuală clasele conţin proprietăţi şi metode care pot fi reutilizate. Acest concept se bazează pe semantica obiect conservat.
- Schemele externe nu pot fi subscheme ale schemei conceptuale [Berti 92], [Guerr 97], iar schemele externe, pot include clase care nu sunt incluse în schema conceptuală. Integrarea este făcută direct în schema externă folosind relaţia derivată de tip "may-be". Acest concept combină semanticile obiect generat şi obiect rezervat. Avantăjul acestui concept constă în faptul că schema conceptuală este mai simplă pentru că acele clase intermediare fără înţeles nu sunt generate. Folosirea relaţiei derivate în schema externă, extinde paradigma orientată obiect, pentru că schemele externe folosite în aplicaţiile program nu sunt relaţii de tip "is-a".

Aşa cum s-a arătat în capitolul 2 (paragraf 2.8.1), ODMG 2.0 permite definirea interogărilor, ele având OQL (Object Query Language) ca limbaj de

programare. Folosind acest mecanism de definire al claselor derivate, lucrurile pot fi interpretate similar cu folosirea interogărilor pentru definirea vederilor. Datorită acestui fapt pornind de la standardul ODMG, în mediile relațional obiectuale, prin definirea claselor vederi sunt definite de fapt clase derivate. Ele permit definirea noțiunilor de bază pentru clasa derivată. Faptul că într-un mediu relațional obiectual putem defini o vedere ca şi o clasă, căreia să-i ataşăm proprietăți şi metode, va permite includerea ei în schema conceptuală şi în schemele externe. Generarea se va face, fie manual, fie automat, folosind diverse medii soft specializate.

Într-un mediu relațional obiectual se folosesc:

- Standardul ODMG pentru a include noile concepte;
- Extinderea limbajului ODL pentru definirea schemei externe;
- Declararea vederilor ca şi clase, pentru a folosi avantajele modelului obiectual;
- Dezvoltarea unor algoritmi pentru generarea schemei externe din setul de clase existent în dicționarul de clase.

Definirea claselor derivate se face prin relația de moştenire ele fiind memorate în dicționarul de date ca şi clase ale unui tip de bază.

Limbajul pentru definirea schemei externe poate avea comenzi predefinite specifice limbajului de tipul "Define class", "Modify class", "Delete class", "Add class", prin care se crează schema externă virtuală. Poate exista un limbaj specific cu comenzi specifice:

- *Define schemaex* pentru crearea schemei externe;
- *Modify schemaex* pentru modificarea schemei externe;
- *Add class to schemaex* adăugarea de clase la schema externă;
- *Define relation between class* definirea relațiilor între clase.

Din cele prezentate anterior în acest capitol rezultă că în procesul de elaborare a schemei externe sunt folosite informații din dicționarul de date, algebra obiectuală, funcții de subsumare şi închiderea moştenirii.

Schemele externe şi clasele derivate pot fi generate şi în bazele de date care respectă standardele ODMG, cu toate că acest lucru nu este prevăzut. Procesul de generare trebuie să respecte algoritmi individuali care iau în considerare dicționarul de date, de unde sunt definite anumite relații de moştenire între clase. Unele relații de moştenire, însă nu pot fi obținute din dicționarul de date şi de aceea este necesară folosirea algebrei obiectuale care prin operatorii select, project, union etc, permit definirea relațiilor de moştenire dintre clasele derivate şi clasele de bază. Dacă cele două metode enunțate anterior nu permit definirea tuturor claselor derivabile, pot fi folosite funcții boolene de subsumare între clase, care determină dacă între respectivele clase există relația de moştenire [Runde 92-2]. Dar conform [Schmo 83] nici acest lucru nu este foarte relevant, deşi prin aceasta pot fi evitate o parte din restricțiile impuse de către definițiile predicatelor care apar în expresiile

funcțiilor. Închiderea moştenirii se bazează pe mecanismul propus în [Motsc 96], pentru clasificarea automată a tipurilor. Ea cere ca pentru fiecare pereche de clase a schemei trebuie să existe o a treia clasă în schemă care conține intersecția tipurilor celor două clase şi uniunea extensiilor celor două clase. Închiderea moştenirii este un proces care face uşor procesul de generare a schemei externe. Oricum prin acest proces, este generată o nouă clasă, care conține tipurile comune ale perechii claselor şi uniunea extensiilor claselor, care este transparentă pentru un utilizator obişnuit.

5.7 Concluzii privind schemele externe

Schemele externe pot conține clase derivate definite prin semanticile obiect conservat şi obiect generat.

Clasele derivate oferă o nouă interfață pentru accesul la informația existentă în clasele de bază. În evoluția schemei considerăm că este necesară definirea claselor parțial derivate, adică a claselor ce conțin elemente nederivabile. Dacă unele clase cu informație nederivabilă trebuie să fie incorporate în scheme externe, informația nederivabilă va fi inclusă în schema conceptuală. Unele operații de derivare prezentate în capitolul 5 pot fi folosite pentru a produce clase parțial derivabile.

În cele prezentate clasele parțial derivate pot conține elemente nederivabile atât în grad cât şi în extensie.

În scopul de a permite modificări neprevăzute a fost definit conceptul de schemă externă temporală. Când o schemă externă cu informații nederivate este definită, schema conceptuală va fi modificată în scopul de a include informație nederivată în noua schemă.

Definirea schemelor externe este un proces laborios în care trebuie determinate toate relațiile de moştenire dintre clase. În acest scop trebuie folosite dicționarele de date, algebra obiectual booleană, închiderea moştenirii, precum şi funcții de subsumare. Prin faptul că mediile relațional obiectuale permit folosirea acestor concepte şi respectă standardele actuale, elaborarea aplicației s-a făcut într-un astfel de mediu soft. Noul standard ODMG cu toate că nu are prevăzut mecanisme de generare a schemelor externe, permite acest lucru. Cei doi algoritmi prezentați în captolul 5, permit realizarea schemelor externe în OODBMS sau RODBMS.

6

Sisteme de fabricaţie. Sisteme de fabricaţie a sculelor

Apariţia unor noi tehnologii software datorită progresului în informatică, permite crearea de sisteme de programe cu interfeţe om/maşină mai facile. În epoca noastră, marcată pe profunde schimbări tehnice şi tehnologice, sistemele de fabricaţie apar ca o rezultantă firească, prin îmbinarea tehnologiilor de vârf cu tehnica de calcul şi cu forţa de muncă superior calificată. Dezvoltarea explozivă a mediului WEB în ultimii ani, determină schimbarea modului de proiectare şi realizare a sistemelor de gestiune, prin corelarea acestora cu noile modele ale bazelor de date şi cu noile tehnologii WEB.

6.1 Sisteme de fabricaţie

Atât componentele hardware cât şi componentele software ale unui sistem de fabricaţie pot fi separate în blocuri funcţionale, putând fi privit ca un ansamblu integrat de maşini cu comandă numerică deservite de un sistem automatizat de manipulare, transport şi depozitare a semifabricatelor, pieselor finite, sculelor şi dispozitivelor, prevăzut cu mijloace automatizate de măsurare şi testare, capabil să realizeze sub comanda calculatorului fabricarea simultană sau succesivă a unor piese de tip diferit aparţinând unei anumite familii, în condiţii de intervenţie minimă a operatorului uman şi cu timpi de reglare reduşi.

Creşterea substanţială a cantităţii şi diversităţii produselor, precum şi a cerinţelor şi preferinţelor privind performanţele acestora conduce tot mai mult la mutaţii şi transformări continue în structura proceselor tehnologice. Pe lângă exigenţele obişnuite privind fiabilitatea, precizia şi productivitatea procesului tehnologic, apar din ce în ce mai mult exigenţe legate de economicitatea energiei, integrarea pe o treaptă superioară a omului în procesul de producţie, dar mai ales cele legate de asigurarea unei flexibilităţi ridicate. Aceste cerinţe au impus, şi impun în continuare producătorilor de bunuri materiale, acceptarea unei noi atitudini şi filosofii faţă de automatizarea sistemelor tehnologice de

fabricație tradiționale. Elementul de noutate îl constituie flexibilitatea sau capacitatea de adaptare rapidă şi sigură a sistemelor la o serie de modificări ale tehnologiei de fabricație.

Automatizarea flexibilă a proceselor tehnologice de prelucrare reprezintă în prezent pe plan mondial pilonul central al procesului de obținere a sistemelor integrate de producție, caracterizate prin asocierea unor echipamente şi utilaje complexe cu sisteme informatice de înaltă performanță, asamblând într-o viziune ierarhică unitară funcțiile de control, manipulare, transport, depozitare alături de funcția de prelucrare.

Procesul de automatizare, care se dezvoltă tot mai puternic, se caracterizează prin creşterea substanțială, în structura utilajelor industriale moderne, a ponderii elementelor electronice şi a softului. În acest sens, s-a ajuns ca ponderea părții mecanice a utilajelor, în expresie valorică, să fie de aproximativ 25%.

Practica a demonstrat din ce în ce mai mult că operatorul uman, prin natura activității pe care o desfăşoară în fața calculatoarelor şi intervenind inteligent mai ales în adaptarea procesului tehnologic la caracterul de unicat sau de serie mică al producției, nu mai poate influența hotărâtor asupra preciziei şi productivității sistemului tehnologic.

Micşorarea rapidă a duratei de viață a produselor ca şi a tehnologiilor de prelucrare, costul din ce în ce mai ridicat al activității de soft în raport cu hardul, exigențele sporite în domeniul calității şi preciziei de prelucrare, utilizarea pe scară largă a controlului automat activ, dificultățile de acoperire cu operatori umani a celui de-al treilea schimb al zilei de lucru, faptul că aproximativ 70-80% din volumul producției are caracter de serie mică sau unicate constituie argumente decisive la tendințele ce se manifestă pe plan mondial cu privire la concepția şi exigențele impuse sistemelor tehnologice de prelucrare: creşterea productivității şi a calității acestora prin mărirea flexibilității. fiabilității şi diminuarea sau chiar eliminarea intervenției operatorului uman din activitățile de execuție.

6.2 Sisteme flexibile de fabricație

Sistemul Flexibil de Fabricație – SFF sau **FMS - Flexible Manufacturing System** se defineşte diferit de la o țară la alta, dar în esență este o unitate de producție capabilă de a fabrica o gamă (familie) de produse discrete cu o intervenție manuală minimă. El cuprinde posturi de lucru echipate cu capacități de producție (maşini-unelte cu comandă numerică sau alte utilaje de asamblare sau tratament) legate printr-un sistem de manipulare a materialelor, în scopul deplasării pieselor de la un post de lucru la altul, funcționînd ca un sistem integrat cu comandă complet programabilă .

Apariţia şi dezvoltarea sistemelor flexibile de prelucrare, asamblare, turnare, sudare au constituit un real sprijin în dezvoltarea sistemelor CIM . Dintre acestea, evoluţia cea mai rapidă au avut-o sistemele flexibile de prelucrare. Această tendinţă s-a remarcat încă de la începutul utilizării maşinilor cu comandă numerică, incluzând dezvoltarea sistemelor DNC până la o treaptă superioară de automatizare a FMS şi încorporarea acestora în CIM.

Fiind sisteme înalt automatizate, sistemele flexibile de fabricaţie, indiferent de natura proceselor comandate, sunt conduse de calculatoare.

În scopul imbunătăţirii modului de funcţionare, pentru a le mări gradul de flexibilitate şi în acelaşi timp productivitatea, aceste sisteme necesită structuri software tot mai evoluate, utilizând din ce în ce mai mult elemente de inteligenţă artificială.

Atât componentele hardware cât şi componentele software ale unui sistem flexibil de fabricaţie (figura 6.1) pot fi separate în blocuri funcţionale, FMS putând fi privit ca un ansamblu integrat de maşini cu comandă numerică deservite de un sistem automatizat de manipulare, transport şi depozitare a semifabricatelor, pieselor finite, sculelor şi dispozitivelor, prevăzut cu mijloace automatizate de măsurare şi testare, capabil să realizeze sub comanda calculatorului fabricarea simultană sau succesivă a unor piese de tip diferit aparţinând unei anumite familii, în condiţii de intervenţie minimă a operatorului uman şi cu timpi de reglare reduşi.

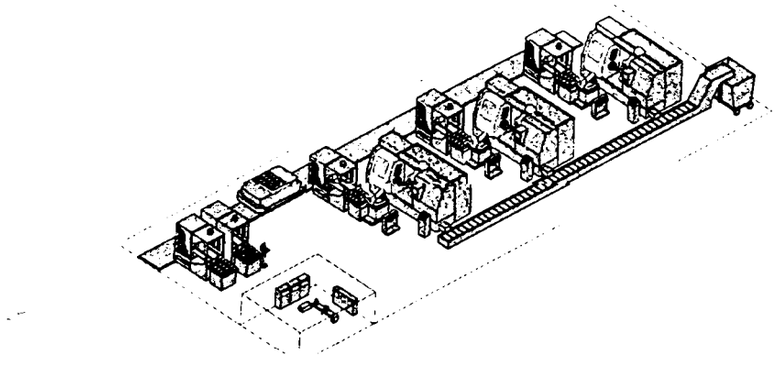


Fig 6.1 Sistem flexibil de fabricaţie

În metodica de concepţie a unui SFF se disting, în general, următorii paşi:

- *stabilirea şi analiza sortimentului de piese şi a programelor de producţie, precum şi a proceselor tehnologice;*
- *separarea grupelor de piese după criteriile tehnologiei de grup, precum şi analiza influenţei acestei împărţiri asupra numărului de utilaje;*
- *stabilirea caietului de sarcini de producţie a sistemului flexibil de fabricaţie pe un interval de timp, precum şi a sarcinilor suplimentare, a modificărilor tehnologice şi a sortimentului de producţie;*
- *elaborarea tehnologiei produselor, ţinându-se cont de condiţiile*

specifice de realizare în cadrul SFF;

- *stabilirea preliminară a cerinţelor pentru subsistemele funcţionale;*
- *determinarea nivelului de automatizare şi integrare a sistemului de prelucrare;*
- *stabilirea variantelor de configuraţii;*
- *elaborarea variantelor alcătuirii subsistemelor de prelucrare prin determinarea tipurilor de maşini-unelte precum şi a celorlalte instalaţii;*
- *dimensionarea preliminară a variantelor SFF;*
- *alegerea variantelor sistemului de prelucrare;*
- *dimensionarea finală a subsistemului de prelucrare pentru varianta aleasă;*
- *stabilirea finală a cerinţelor în raport cu subsistemele funcţionale;*
- *stabilirea tipurilor de subsisteme de transport şi înmagazinare;*
- *dimensionarea preliminară a subsistemelor de transport şi înmagazinare;*
- *alegerea variantei în dimensionarea finală a subsistemului de transport şi înmagazinare;*
- *proiectarea subsistemului de manipulare;*
- *proiectarea altor subsisteme de derulare a fluxurilor materiale şi energetice;*
- *identificarea circuitului de informaţii în SFF;*
- *proiectarea arhitecturii aparaturii de automatizare şi calcul.*
- *Concepţia unui sistem flexibil de fabricaţie nu se realizează în general prin parcurgerea secvenţială a ansamblului de module, ci necesită un număr de reluări din diferite puncte de intrare cu modificarea parametrilor, pe baza evaluării rezultatelor obţinute.*

- *Efectele introducerii SFF în intreprinderi sunt multiple, dintre aceste efecte pot fi specificate următoarele:*
- *creşterea productivităţii muncii;*
- *îmbunătăţirea calităţii produselor;*
- *reducerea numărului de maşini-unelte;*
- *reducerea spaţiului de producţie şi a spaţiilor auxiliare;*
- *posibilitatea modificării flexibile a sortimentelor de produse fabricate;*
- *scurtarea timpului de producţie;*
- *reducerea consumului de energie, materii prime şi materiale;*
- *reducerea preţurilor unitare de fabricaţie;*
- *reducerea numărului de angajaţi;*
- *reducerea poluării;*
- *creşterea randamentului în conducerea intreprinderii.*
- *Din punct de vedere al funcţiei principale a SFF, reprezentată de producţia realizată, se pot deosebi trei grupe de activităţi:*
- *activităţi de conducere, reglare, comandă şi organizatorice – legate de organizarea procesului de producţie şi încărcarea instalaţiilor şi care cuprinde:*
- *elaborarea şi verificarea planurilor operaţionale;*
- *ordonarea culegerii de date;*
- *prelucrarea datelor;*
- *comanda maşinilor şi a celorlalte elemente, a procesului şi al schimbului de informaţii;*
- *încărcarea magaziilor de piese;*
- *încărcarea magazinelor de scule aşchietoare;*
- *sinteza şi transmiterea programelor de prelucrare şi a programelor de comandă;*
- *încărcarea dispozitivelor de măsură şi control;*
- *reglarea şi codificarea sculelor.*
- *activităţi de diagnosticare, măsurare şi control dimensional – referitoare la piesă, procesul de prelucrare, instalaţii pentru*

asigurarea calităţii de prelucrare cerute;

- *activităţi de manipulare-prelucrare – legată de trecerea pieselor prin sistem şi care presupune:*
- *pregătirea pieselor pentru prelucrare;*
- *transportul pieselor la şi de la maşina de prelucrat, la şi de la posturile intermediare de stocare, precum şi la maşinile de spălat şi de măsurat şi control dimensional;*
- *aşezarea şi fixarea ansamblului paletă-piesă pe masa maşinii-unelte;*
- *ciclul propriu-zis de prelucrare;*
- *prelucrarea pieselor de pe maşina-unealtă;*
- *spălarea, uscarea şi conservarea pieselor după prelucrare;*
- *îndepărtarea deşeurilor tehnologice*

Fiind sisteme înalt automatizate, indiferent de natura proceselor comandate, SFF sunt conduse de calculatoare electronice. În scopul îmbunătăţirii modului de funcţionare, pentru a le mări gradul de flexibilitate şi în acelaşi timp productivitatea, aceste sisteme necesită structuri software tot mai evoluate utilizând din ce în ce mai multe elemente de inteligenţă artificială.

6.3 Sisteme de fabricaţei a sculelor

Sculele dispozitivele şi verificatoarele prin nivelul lor tehnic şi prin sistemul de gestiune stabilesc nivelul de calitate şi de eficienţă al societăţii economice. În economia actuală asmilarea unui produs nou poate fi de o importanţă vitală pentru societatea economică. Dar, din timpul total de pregătire al fabricaţiei, partea de SDV-uri reprezintă aproximativ 60..70%, cu variaţii legate de particularităţile produsului. Elaborarea la timp a pregătirii de fabricaţie permite întocmirea la timp a unei oferte concrete, câştigarea unui segment de piaţă şi respectarea termenilor de livrare cu repercursiuni pozitive în încrederea beneficiarilor şi consolidarea firmei.

În situaţia existentă documentaţia care însoţeşte acest proces este foarte mare. Documentaţia privind sculele poate fi raportată la mai multe domenii şi anume:

- tehnică (desene de execuţie, condiţii tehnice, derogări etc.);
- tehnologică (fişă tehnologică, notă de predare, fişe de magazie);
- lansare (capacitate de producţie, plan de producţie);
- urmărire (comenzi, grafice, fişe de urmărire, note de predare, etc);
- raportări, centralizatoare;
- relansare;
- recondiţionare;
- plan de scule standardizate.

Softul de gestiune trebuie interfaţat şi integrat cu softul care deserveşte în ansamblu sistemul de fabricaţie al sculelor. El trebuie integrat cu celelalte sisteme ale CIM-ului ca de exemplu subsistemul CAD, CAM.

Pe plan mondial softul existent în domeniul sculelor este orientat pe trei direcții și anume:

- de prereglare și gestiune a stocurilor (Mini-Fal, TMS-95, ToolBrain);
- de gestiune a stocurilor, planificare și urmărire a ansamblelor de scule (Corotas, Logofix, Toolware, Surcouf etc);
- softuri integrate în sistemele flexibile de fabricație oferite în special de piața germană și italiană.

Aceste softuri sunt în general dificil de achiziționat datorită prețului ridicat și trebuie adaptate la cerințele concrete ale societății economice din Romania. Un astfel de soft este necesar să fie adaptat și cu un modul de proiectare, deoarece un număr mare de societăți își produc în mare parte necesarul de scule de care are nevoie.

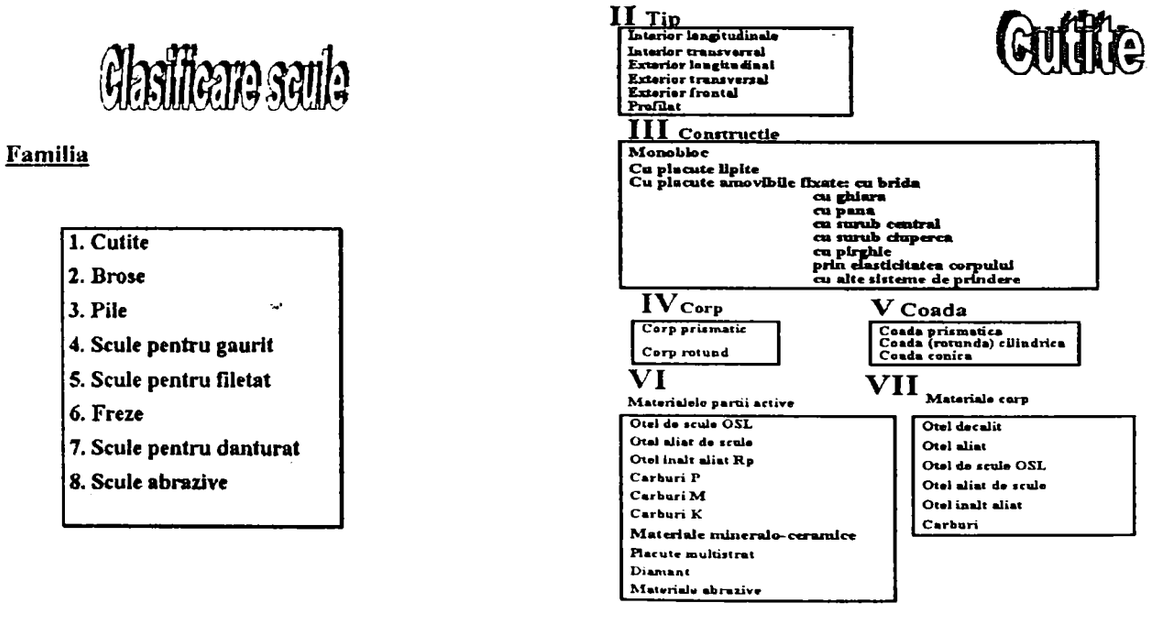
Obiectivele unui astfel de soft implementat într-o intreprindere din țara noastră sunt următoarele:

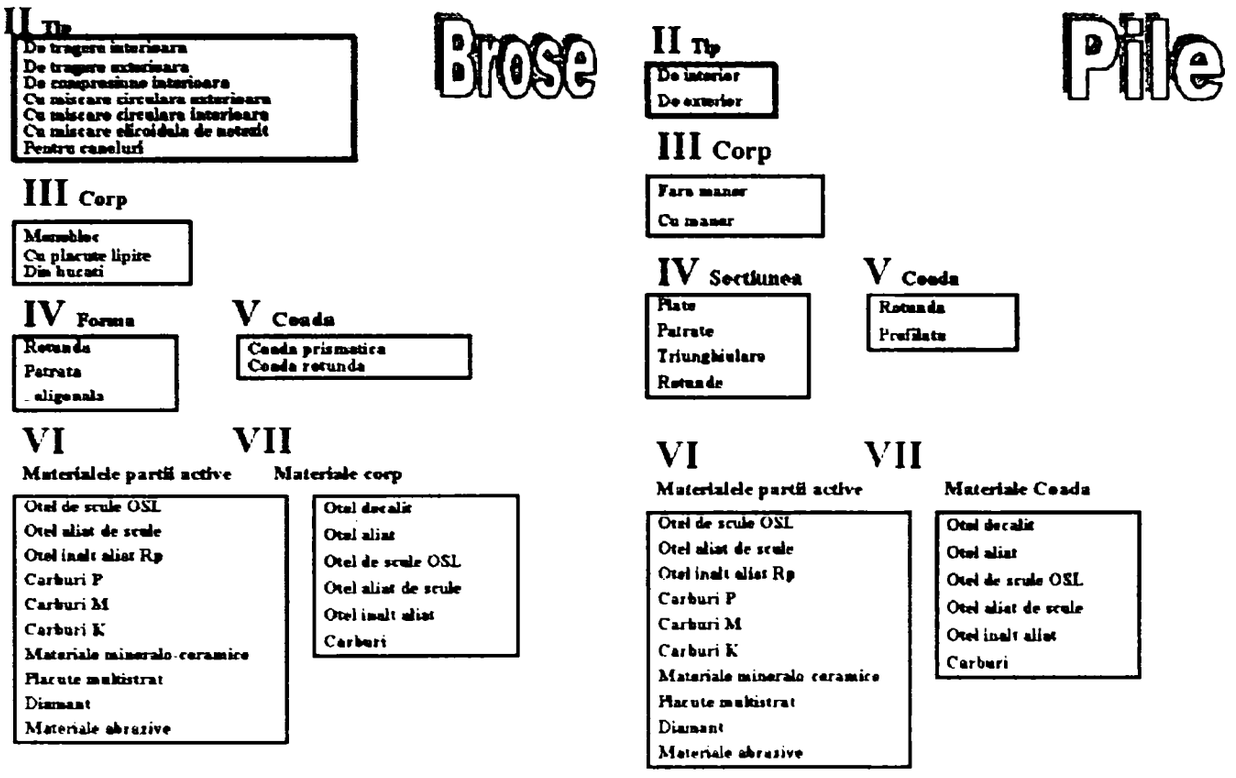
- simplificarea circuitului documentației;
- evitarea reproiectării unor scule;
- cunoașterea exactă a stocurilor în orice moment;
- pregătirea mai rapidă a fabricației;
- reducerea stocului de scule;
- reducerea termenelor de livrare;
- calificarea mai înaltă a întreprinderii (conform ISO 9000)

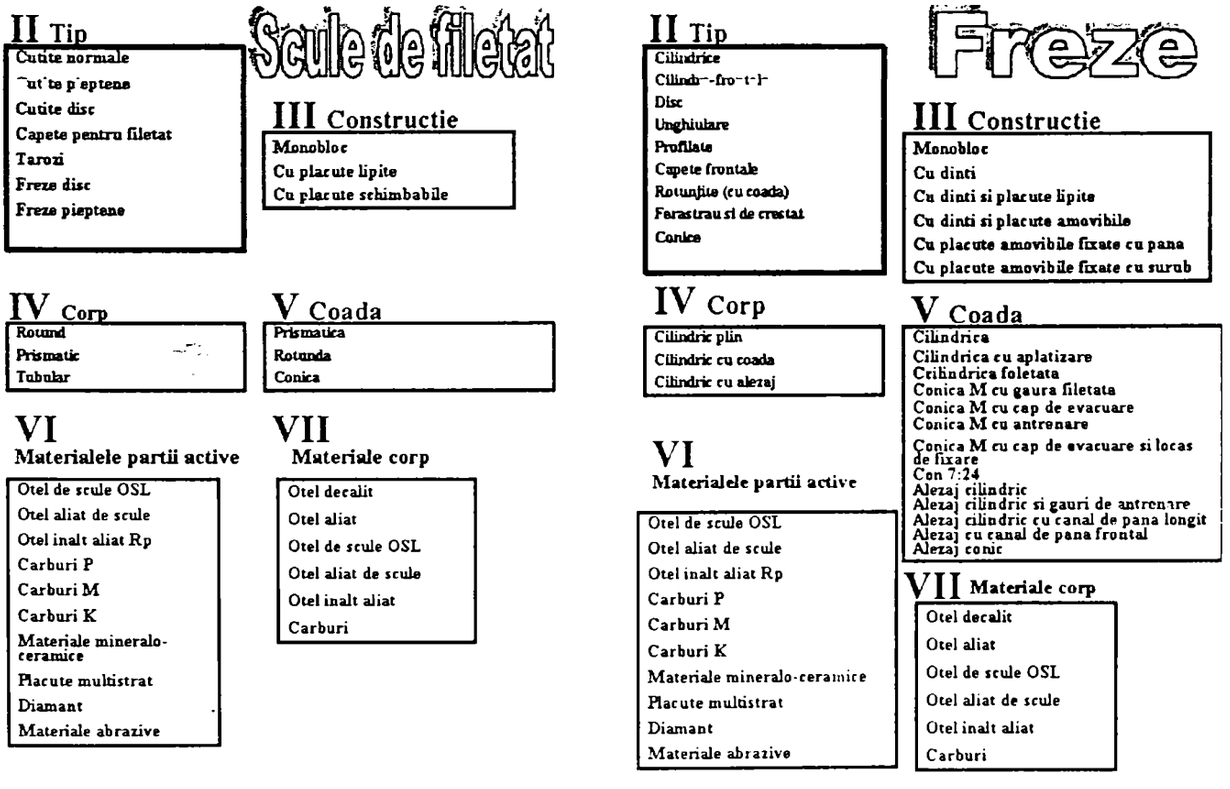
Pentru realizarea unui astfel de soft este necesară realizarea uneia sau a mai multor baze de date care să conțină datele necesare pentru implementarea sa.

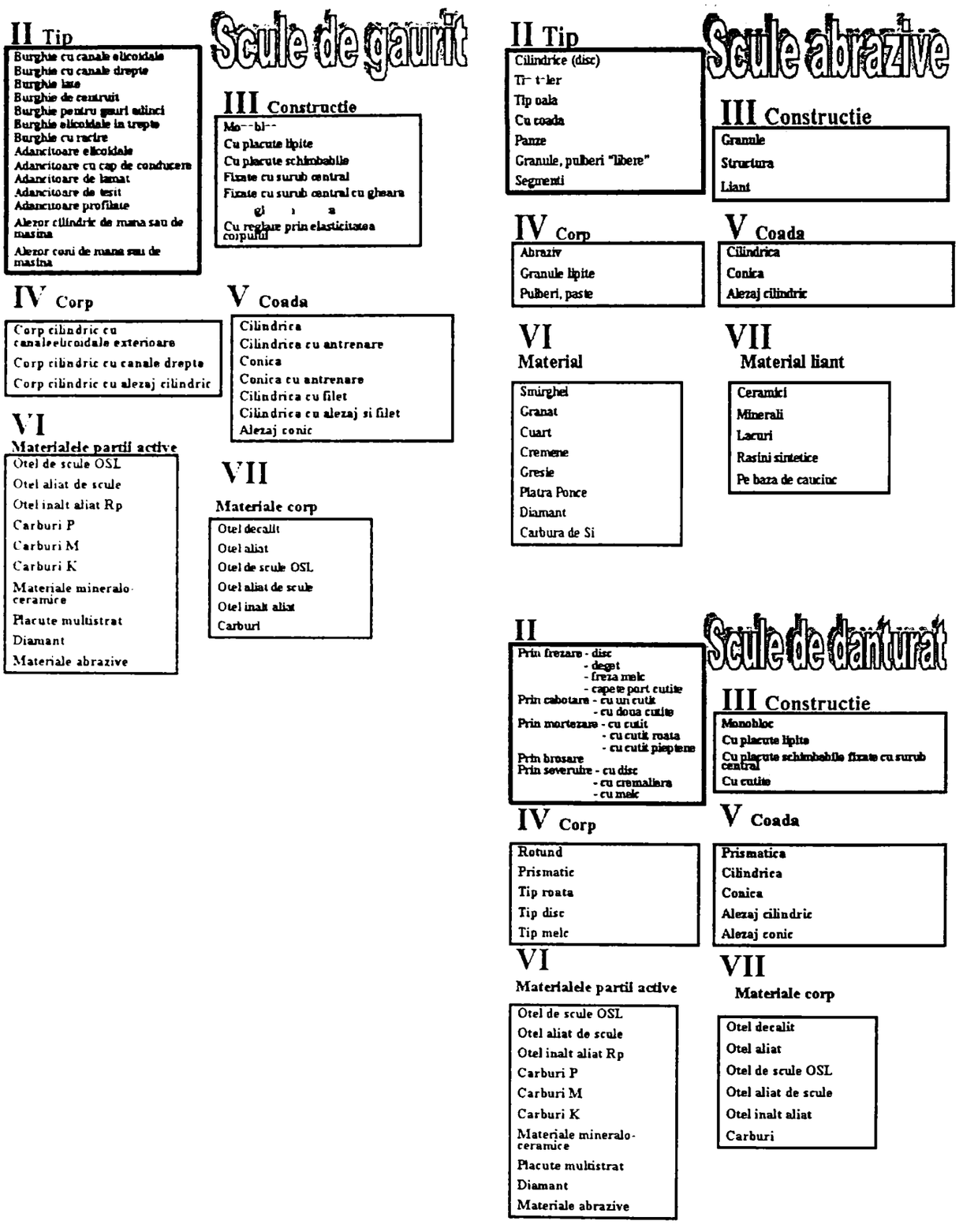
6.3.1 Clasificarea sistemului de fabricație al sculelor

Pentru proiectarea sistemului de fabricație al sculelor s-a pornit de următoarea clasificare a acestora:




II Tip
De tragere interioara
De tragere exterioara
De compresiune interioara
Cu miscare circulara interioara
Cu miscare circulara interioara
Cu miscare elicoidala de netezit
Pentru caneluri
Brose
III Corp
Monobloc
Cu placute lipite
Din bucati
IV Forma
Rotunda
Patrata
Poligonala
V Coada
Coada prismatica
Coada rotunda
VI
Materialele partii active
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat Rp
Carburi P
Carburi M
Carburi K
Materiale mineralo-ceramice
Placute multistrat
Diamant
Materiale abrazive
VII
Materiale corp
Otel decalit
Otel aliat
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat
Carburi
II Tip
De interior
De exterior
Pile
III Corp
Fara maner
Cu maner
IV Sectiunea
Plate
Patrate
Triunghiulare
Rotunde
V Coada
Rotunda
Profilata
VI
Materialele partii active
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat Rp
Carburi P
Carburi M
Carburi K
Materiale mineralo-ceramice
Placute multistrat
Diamant
Materiale abrazive
VII
Materiale Coada
Otel decalit
Otel aliat
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat
Carburi


II Tip
Cutite normale
Cutite pieptene
Cutite disc
Capete pentru filetat
Tarozi
Freze disc
Freze pieptene
Scule de filetat
III Constructie
Monobloc
Cu placute lipite
Cu placute schimbabile
IV Corp
Rotund
Prismatic
Tubular
V Coada
Prismatica
Rotunda
Conica
VI
Materialele partii active
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat Rp
Carburi P
Carburi M
Carburi K
Materiale mineralo-ceramice
Placute multistrat
Diamant
Materiale abrazive
VII
Materiale corp
Otel decalit
Otel aliat
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat
Carburi
II Tip
Cilindrice
Cilindr--fro-t-l-
Disc
Unghiulare
Profilate
Capete frontale
Rotunjite (cu coada)
Ferastrau si de crestat
Conice
Freze
III Constructie
Monobloc
Cu dinti
Cu dinti si placute lipite
Cu dinti si placute amovibile
Cu placute amovibile fixate cu pana
Cu placute amovibile fixate cu surub
IV Corp
Cilindric plin
Cilindric cu coada
Cilindric cu alezaj
V Coada
Cilindrica
Cilindrica cu aplatizare
Cilindrica foletata
Conica M cu gaura filetata
Conica M cu cap de evacuare
Conica M cu antrenare
Conica M cu cap de evacuare si locas de fixare
Con 7:24
Alezaj cilindric
Alezaj cilindric si gauri de antrenare
Alezaj cilindric cu canal de pana longit
Alezaj cu canal de pana frontal
Alezaj conic
VI
Materialele partii active
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat Rp
Carburi P
Carburi M
Carburi K
Materiale mineralo-ceramice
Placute multistrat
Diamant
Materiale abrazive
VII Materiale corp
Otel decalit
Otel aliat
Otel de scule OSL
Otel aliat de scule
Otel inalt aliat
Carburi



6.3.2 Schema bloc a pachetului de programe, care deserveşte sistemul de fabricaţei al sculelor

În figura 6.2 este prezentată schema bloc a pachetului de programe, pentru sistemul de fabricaţie al sculelor.

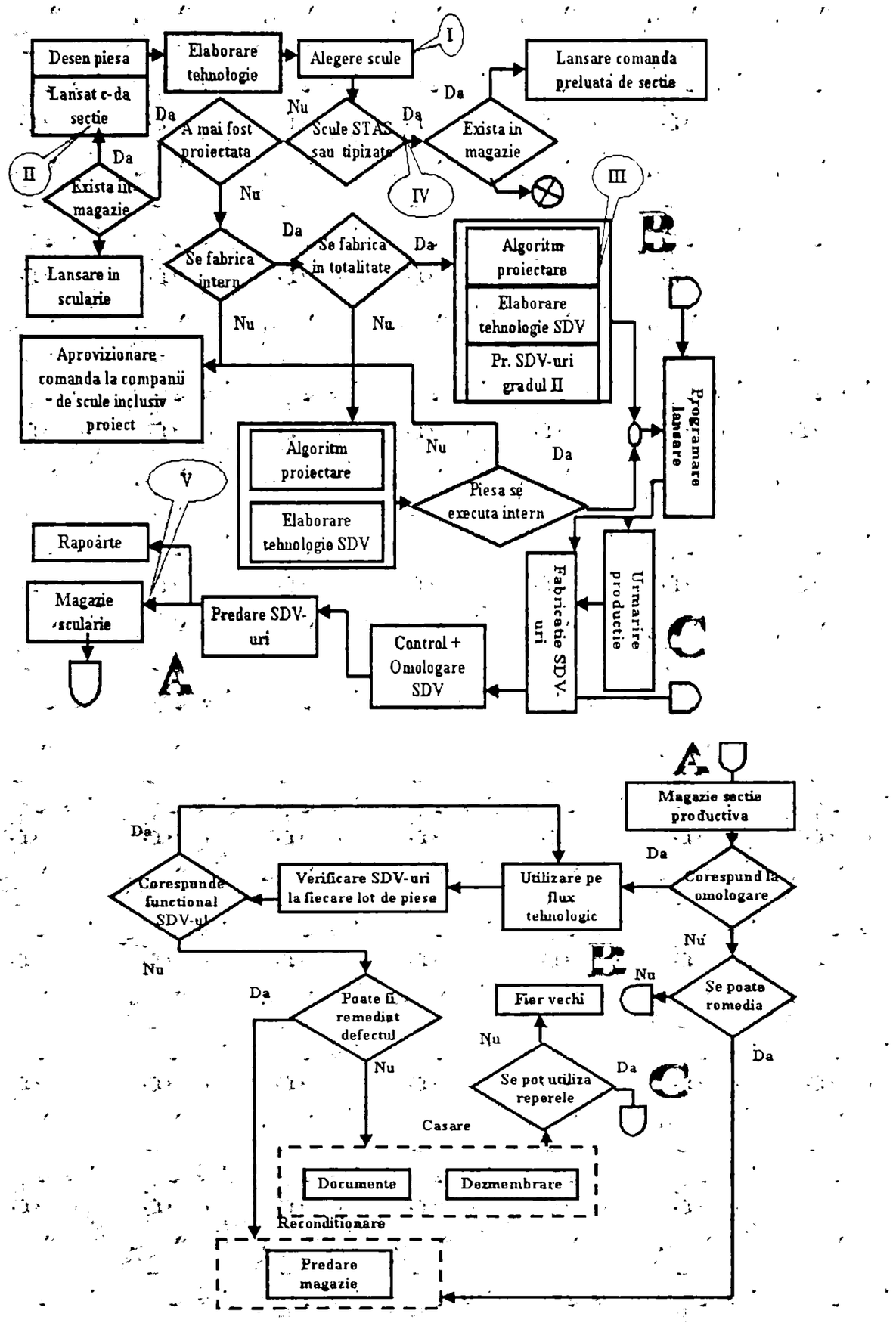


Fig 6.2 Schema bloc a sistemului de fabricație al sculelor

Presupunem că tehnologul de produs trebuie să întocmească lista de scule aferente unei anumite tehnologii de realizare a unei piese. După selectarea sculei dorite, se caută în fişa de magazie dacă scula există. Dacă da se lansează comanda de scoatere din magazie şi trimitere spre secţie. Dacă fişa nu există în magazie se verifică dacă aceasta este standardizată sau nu. Dacă da, se lanseaza comanda spre aprovizionare. Dacă nu , deci este o sculă specială, se verifică dacă a mai fost proiectată.În acest caz în fişă va exista numărul desenului şi se poate face automat lansarea în execuţie, în secţia sculărie. Dacă scula nu a mai fost proiectată este chemată automat subrutina de proiectare constuctivă. Se obţine astfel desenul de execuţiei al sculei care va fi trimis la sculărie.

Activităţile şi documentele care reglementează realizarea şi circulaţia sculelor sunt următoarele:

1. *Proiectarea sculelor* are următoarele activităţi caracteristice:
- alegerea SDV-urilor standardizate;
- proiectarea SDV-urilor speciale;
- documente caracteristice:
- desene de execuţie scule;
- desene de ansamblu scule;
- cerere de execuţie pentru SDV-uri speciale.

2. *Planificare Lansarea producţiei* are următoarele activităţi caracteristice:
- determinarea capacităţilor de producţie ale secţiei sculărie;
- stabilirea consumurilor specifice de scule standardizate şi speciale;
- stabilirea necesarului de aprovizionat cu scule standardizate şi speciale;
- stabilirea necesarulu de materii prime şi materiale pentru scule.
- documente caracteristice:
- centralizatoare privind capacităţîle de producţie, privind consumurile specifice al sculelor STAS, fişe de consum specifice pe reper al SDV-urilor speciale, liste de aprovizionat: scule speciale, scule standardizate, materii prime şi materiale, lista consumurilor specifice de materiale pentru scule.

3. *Urmărirea producţiei* are următoarele activităţi caracteristice:
- realizarea producţiei ca sortiment, volum ore normă, termene de execuţie;
- compararea şi analiza costurilor;
- stabilirea pe baza normativelor de durabilitate, uzură şi a planurilor de operaţii a normelor tehnice de consum pentru SDV-uri;
- elaborarea de instucţiuni privind transportul, depozitarea şi exploatarea corectă a sculelor;
- verificarea respectării stocurilor minime şi maxime se scule precum şi verificarea lor.
- documente caracteristice:
- fişă de evidenţă scule;
- raport zilniv de livrări;
- fişă de magazie pentru scule standardizate;
- note de transfer;
- note de sesizări

4. *Tenolog sculărie* are următoarele activităţi caracteristice:
- elaborarea tehnologiei de execuţie a reperelor şi de asamblare a sculelor;
- arhivarea documentaţiei tehnice;
- realizarea unor tehnologii noi de fabricare şi întreţinere a sculelor;
- elaborarea documentaţiei constructiv tehnologice pentru recuperarea prin recondiţionare a sculelor normalizate şi standardizate.
- documente caracteristice:

- fişa tehnologică;
- bon de materiale;
- bon de lucru.

5. *Producția* are următoarele activități caracteristice:
- înregistrarea sculelor speciale noi în fişele de magazie;
- asigurarea materialelor necesare realizării producției de scule;
- propunerea de norme de consum pentru regie ale sculăriei;
- asigurarea condițiilor de încercare şi omologare a sculelor noi odată cu produsele.
- documente caracteristice:
- fişe de magazie pentru SDV-uri speciale;
- note de transfer pentru materiale;
- bonuri de consum – scule uzate;
- note de predare.

6. *Evidența sculelor* are:
- activități caracteristice:
- depozitarea şi eliberarea sculelor speciale şi controlul lor tehnic la înapoierea din producție;
- menținerea sculelor în stare de folosință prin trimiterea lor la ascuțire sau reparare;
- recepționarea sculelor standardizate recepționate din exterior;
- depozitarea şi evidența centralizată a lor;
- urmărirea situației stocului de scule existent.
- documente caracteristice:
- fişe de magazie pentru scule speciale şi standardizate;
- fişe de umărire şi evidență a SDV-urilor în exploatare.

7. *Recondiționare şi casare* are următoarele activități caracteristice:
- trimiterea sculelor uzate la recondiționat;
- scoaterea din uz a sculelor neutilizate.
- documente caracteristice:
- fişa de urmărire şi evidență în exploatare;
- fişa de constatare şi executare de lucrări;
- proces verbal pentru scoaterea din uz a obiectelor de inventar date în folosință.

8. *Vânzare* are următoarele activități caracteristice:
- întocmirea facturilor pentru diverse scule vândute;
- arhivarea comenzilor şi a facturilor emise şi încasate la nivel de secție sculărie;
- emiterea de situații către compartimentele financiar contabile ale societății.
- documente caracteristice:
- facturi şi comenzi de evidență a vânzărilor şi a încasărilor ca urmare a activității de vânzare cumpărare a sculelor;
- note contabile privind activitatea de vânzare cumpărare

Toate aceste activități vor fi realizate într-o aplicație care va deservi sistemul de fabricație a sculelor într-o societate economică. Acest sistem de fabricație poate fi împărțit în mai multe subsisteme funcție de activitățile enumerate. Datele existente pot fi memorate funcție de diverse structuri de date. Pentru fiecare structură de date, poate fi implementată o bază de date specifică.

Din punct de vedere hard sistemul este implementat înr-o rețea locală de tip INTRANET la nivel de societate economică (secția sculărie) legată eventual într-o rețea INTERNET.

În figura 6.3 este reprezentat suportul hardware, pe care este implementat software-ul, destinat sistemului de fabricație al sculelor.

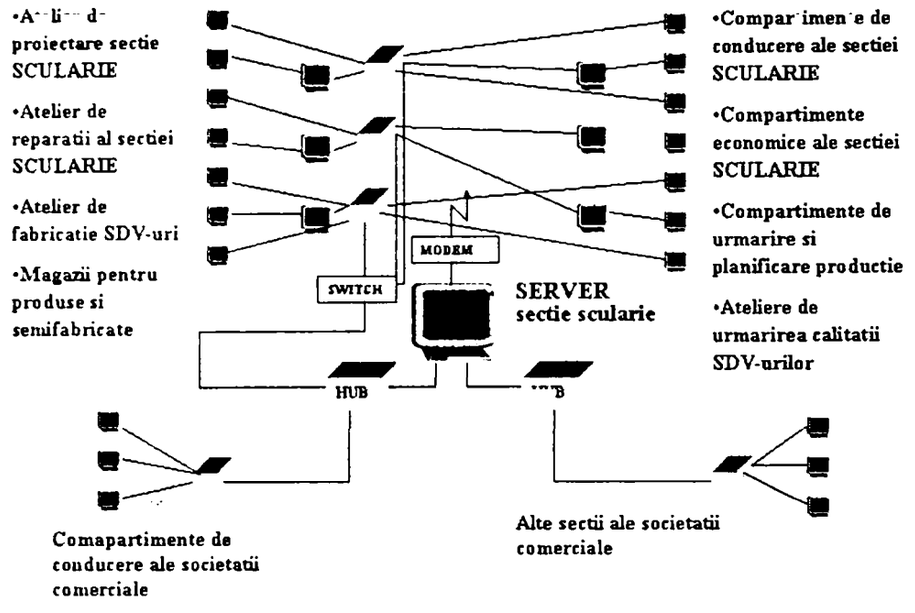


Fig 6.3

6.4 Definirea modelului conceptual pentru sistemul de fabricație al sculelor într-o bază de date orientată obiect

Problema realizării unui prototip pentru SFS într-un mediu obiectual, este o problemă complexă, care implică o serie de aspecte legate de mediul în care va fi implementat, el trebuind să respecte paradigma obiectuală. Pentru implementare s-au ales un mediu pur obiectual, Jasmine CA şi un mediu relațional obiectual Visual Fox. O astfel de implementare, după cunoştințele mele nu a mai fost realizată. Cele două implementări vor reliefa existența claselor derivate şi a schemelor externe precum şi particularitățile de realizare practică a SFS-ului.

Modelul conceptual este format din două părți şi anume modelul operațional, care se referă în special la partea de control a sistemului şi modelul deductiv format din: predicate de bază, predicate derivate, constrângeri de integritate şi codiții de ieşire a sistemului [Olivé 89].

Conform [Date 02] în literatura de specialitate termenul de model de date are două înțelesuri:

- ca un limbaj de programare în care constructorii sunt folosiți pentru a rezolva probleme complexe;

- este un anume program scris într-un limbaj de programare sau cu alte cuvinte este modelul datelor persistente, pentru diverse stări particulare ale sistemului.

În particular un model de date este o construcţie, care ilustrează aspectele concrete ale datelor. Conform [Date 02], un model de date bun este acela care ilustrează aspectele logice ale datelor şi mai puţin aspectele fizice. Modelul va include un set de obiecte, împreună cu un set de operatori care vor executa operaţii asupra obiectelor. Putem considera dintr-un punct de vedere că obiectele şi operatorii constitue ceva abstract. Implementarea modelului este realizarea fizică a părţii abstracte. Modelele obiectuale îşi au originea în retelele semantice si în limbajele de programare orientate pe obiecte. Cerinţele unui model obiectual pentru SFS au fost prezentate în cap. 3 paragraf 3.10. Obiectivul principal este de a prezenta mai bine realitatea si de a beneficia de reutilizarea obiectelor, pentru a construi obiecte din ce în ce mai complexe. În abordările traditionale, modelele de reprezentare sunt foarte diferite de-a lungul etapelor ciclului de dezvoltare a sistemului informatic. În activitatea de modelare obiectuală, conceptul de bază este *obiectul*, pe care îl putem defini ca fiind reprezentarea unei entităţi a lumii reale. Aşa cum s-a prezentat în capitolul 2, în baza de date, obiectul este identificat într-o manieră unică printr-un identificator, numit OID (*Object IDentifier*), atribuit de sistem. Obiectele din lumea realã, de aceeaşi natură sau prezentând similitudini, sunt grupate în cadrul unei *clase,* în care sunt declarate toate atributele precum si metodele care le manipulează. O clasă constituie astfel un tipar pentru o multime de obiecte. În contextul bazelor de date obiectuale, notiunea de clasă de obiecte acoperă un aspect intensional (adică descrierea) şi un aspect extensional (ansamblul de obiecte ale clasei). Orice obiect este construit apoi, plecând de la o clasă (sau un tip) care regrupează structura datelor şi metodele. O clasă joacă astfel rolul unui generator de obiecte. Metodele de creare si de modificare sunt indispensabile tuturor claselor pentru a crea şi a actualiza obiectele clasei. Clasele unei aplicatii sunt organizate sub formă de graf. Se obţine astfel un graf de moştenire, adică un graf unde o clasă moşteneşte de la superclasa sa. Subclasele sunt obţinute prin rafinări succesive ale super-claselor. Este unul dintre aspectele reutilizabilităţii. Există două tipuri de legături într-un graf de clase: legătura de moştenire şi legătura de referinţă. Moştenirea provine din decupajul lumii reale făcut prin mecanismul de specializare. Cât despre legătura de referinţă, ea permite stabilirea unei legături oarecare între două clase, indiferent dacă este vorba de compunere, de agregare, de derivare sau pur si simplu desemnarea unui obiect pornind de la un altul.

Spre deosebire de modelul relaţional, un model obiectual permite nu doar descrierea aspectului static al unei aplicaţii în materie de date şi de structură, ci şi a aspectului dinamic, în materie de comportament al obiectelor şi de comunicare. Astfel, toate procedurile de manipulare a atributelor sunt

proprietatea clasei şi nu sunt cunoscute altor obiecte decât prin semnăturile lor, adică prin numele metodei, prin argumente şi prin natura rezultatului returnat.

6.4.1 Arhitectura modelului conceptual al sistemului de fabricaţie al sculelor

Arhitectura modelului conceptual, pentru Sistemul de Fabricaţie al Sculelor (SFS), conţine schema conceptuală, definirea obiectelor, schemele externe ale sistemului (figura 6.4).

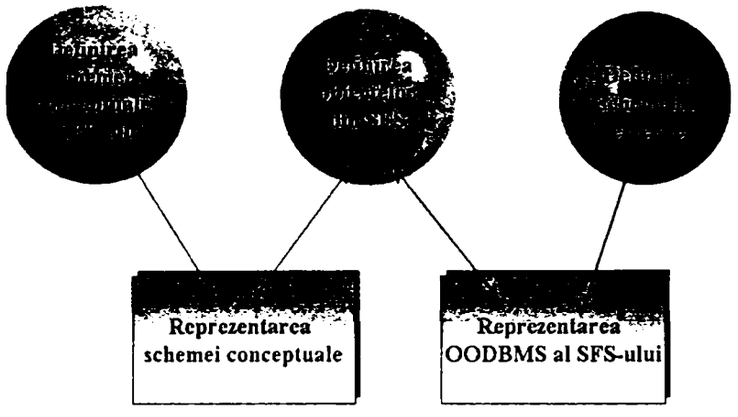


Fig 6.4 Arhitectura SFS-ului

Un exemplu de schema conceptuală pentru o parte a unui sistem de fabricaţie (dezvoltat în Jasmine CA), este prezentată în figura 6.5.

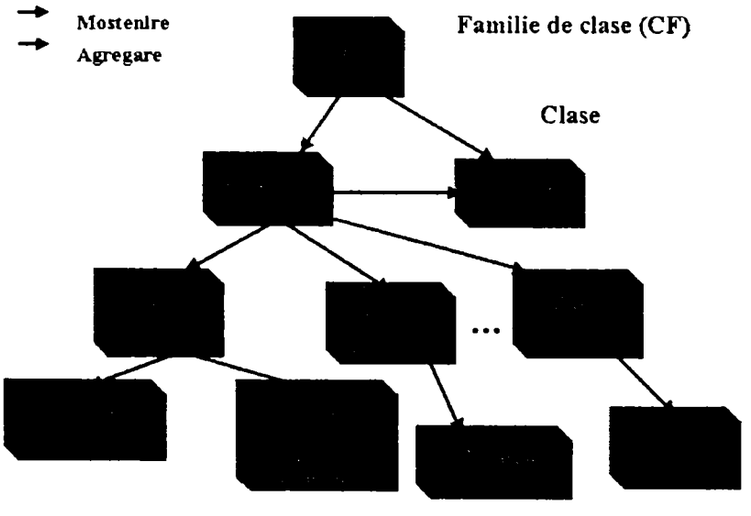


Fig 6.5 Schema conceptuală a SFS-ului

6.4.2 Predicatele modelului obiectual al SFS-ului

Putem considera predicatele de bază ale modelului:

- *definirea claselor* care pot fi clase predefinite sau clase de obiecte utilizator;
- definirea obiectelor în clase;

- definirea proprietăţilor;
- definirea metodelor;
- construirea claselor în cadrul sistemului;

Folosind comenzi specifice şi limbajul ODQL, în Jasmine CA modelul obiectual poate fi realizat astfel:

```
Create store –numberofPages 2000 –pagesize 8
modelscule %jas_system%\jasmine\data\scule
CreateCF sculeCF
DefaultCF sculeCF
DefineClass sculeCF::composite_scule
        Super: systemCF::composite
        Description: "Clasa abstracta de bază
pentru definirea claselor"
DefineClass sculeCF::scule
        Super sculeCF::composite_scule
{
        Maxinstancesize: 8;
        Instance:
                String caracteristici_tehnice;
};
DefineClass freze
        super sculeCF::scule
{
        maxinstancesize: 4;
        instance:
                integer: D;
                integer: d;
                boolean: cautare();
}
DefineClass Freze_cilindro_frontale
        Super: sculeCF::Freze
{
        maxinstancesize: 4;
        instance:
                bag<string>:        numefreza
default: bag{};
                integer: cod_placuta;
}
DefineClass Freze_cilindrice
        Super: Freze
{
        maxinstancesize: 4;
}
DefineClass cutite
        Super: sculeCF::Scule
{
        maxinstancesize: 4;
        instance:
                integer: l1;
}
DefineClass cutite_strung
        Super: cutite
{
        maxinstancesize: 4;
        instance:
                integer: d;
}
DefineClass alezoare
        Super: sculeCF::Scule
{
        maxinstancesize: 4;
        instance:
                integer: D;
                integer d;
}
DefineClass de_mana
        Super: alezoare
{
        maxinstancesize: 4;
        instance:
                string: tip;
                integer l;
}
buildclass scule freze cutite alezoare
freze_cilindro_frontale.ofreze_cil_front
ofreze_cil_front=freze_cilindro_frontale.new()
ofreze_cil_front.nume='Freze cilindro frontale cu
coada_conica'
freze_cilindro_frontale.addProperty(clprop,"D1",
"decimal",precision:=3,scale:=2,  isunique:=false,
ismandatory:=true);
```

Predicatele *super, instance, maxinstancesize* pot fi considerate predicate drivate care oferă informaţii despre starea obiectului modelat în fiecare moment.

6.4.3 Constrângerile de integritate ale modelului obiectual al SFS-ului

Constrângerile de integritate ale modelului se pot baza pe următoarele reguli [Date 02] :

- Un tip nu poate fi considerat supertip pentru el însuşi;

- Nu există supertipuri redundante într-o definiție de obiect;
- O proprietate poate fi definită o singură dată;
- O clasă nu poate fi superclasă pentru ea însuşi;
- Nu există superclase redundante într-o definiție de clasă;
- Între două sau mai multe clase poate fi definită o singură definiție a proprietăților;

6.4.4 Cerințele externe ale modelului obiectual al SFS-ului

Sistemul, trebuie să creeze o mulțime de termeni corecți, ca răspuns la acțiunea predicatelor de bază sau derivate, ale sistemului, în schema conceptuală. În general, cerințele externe ale modelului obiectual, au fost definite în cap3. paragraf 3.11.

7

Implementarea bazei de date obiectuale, pentru Sistemul de Fabricaţie al Sculelor, într-un mediu software, pur obiectual

În momentul actual lumea bazelor de date se împarte în trei mari categorii: baze de date relaţionale, relaţional obiectual şi pur obiectuale. Implementarea într-un mediu pur obiectual generează o serie de probleme în special din punct de vedere al modului în care este memorată informaţia în cadrul obiectelor abstracte de date precum şi din punctul de vedere al gestionării informaţiei.

7.1 Consideraţii generale privind modul de proiectare a bazei de date în mediul pur obiectual

Pornind de la idea că un mediu pur obiectual se apropie cel mai mult de noţiunile teoretice ale bazelor de date orientate obiect, s-a implementat baza de date obiectuală pentru SFS în mediul CA Jasmine. Într-un astfel de mediu dispare noţiunea de relaţie pentru memorarea datelor. Memorarea datelor se realizează în obiecte. Aceasta este entitatea de bază pentru memorarea datelor.

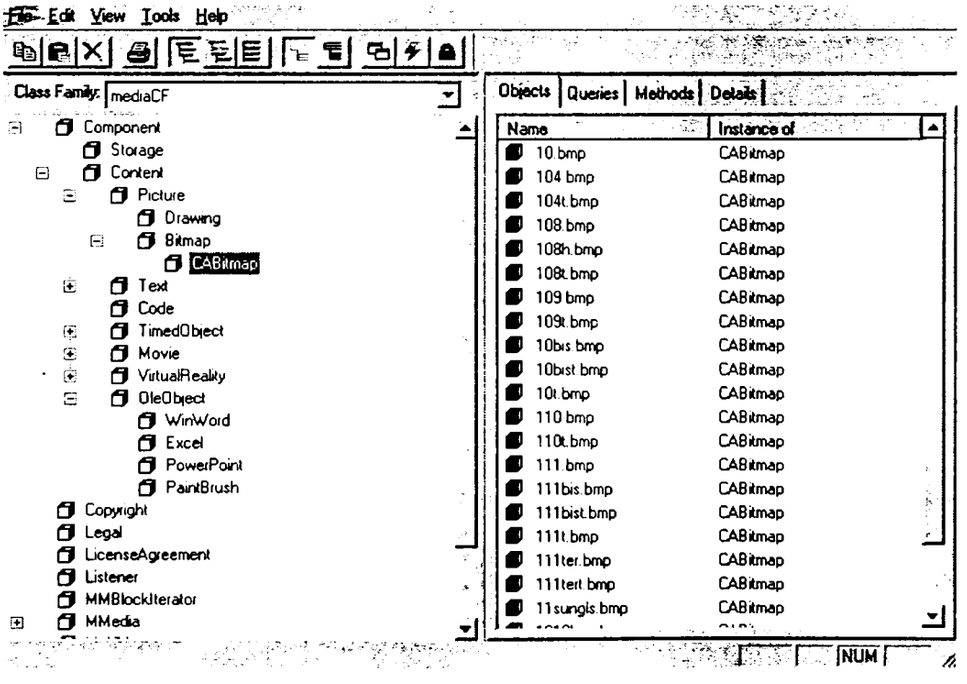


Fig 7.1 Clase predefinite de tip imagine în software-ul Jasmine

Astfel fiecare obiect primeşte un număr de identificare propriu în cadrul bazei de date, şi un nume definit de către utilizator. Până la definirea obiectelor este necesară definirea clară a claselor de obiecte şi a ierarhiei existente între clase. Fiecare entitate existentă în cadrul bazei de date trebuie definită ca un obiect ce aparţine unei clase. Pentru aceasta trebuie definită o clasă cu obiecte de tip documente (text, word, etc), de tip imgini (bmp, gif, tif, etc), clasă cu obiecte specifice mediului Office Windows (Excel, Powerpoint), o clasă cu fişiere imagine, sunet (avi, wav) figura 7.1 şi figura 7.2.

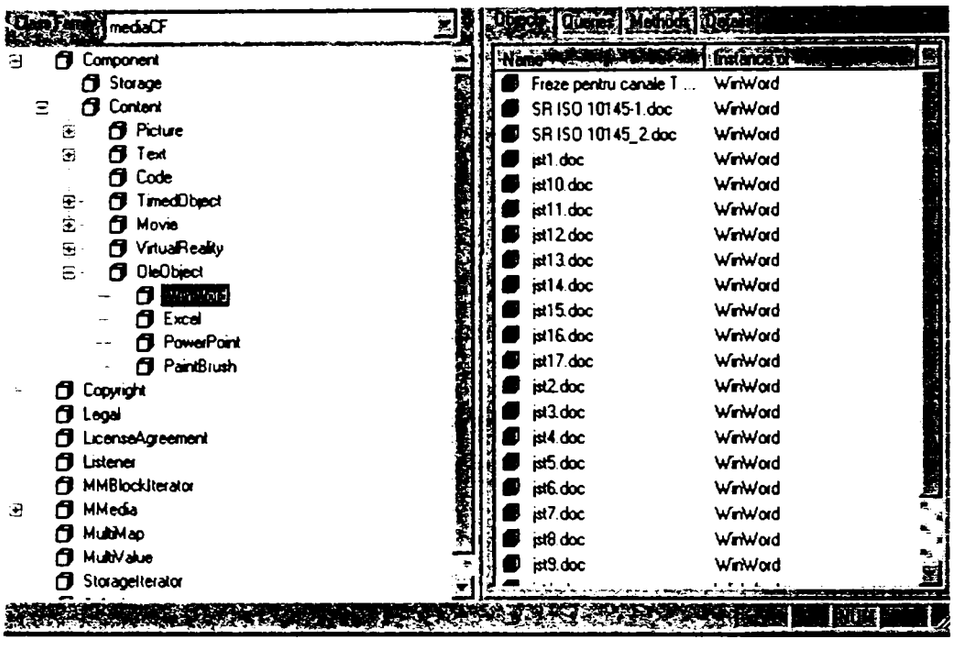


Fig 7.2 Clase predefinite de tip Office în software-ul Jasmine

În general în mediul Jasmine, există două tipuri fundamentale de familii

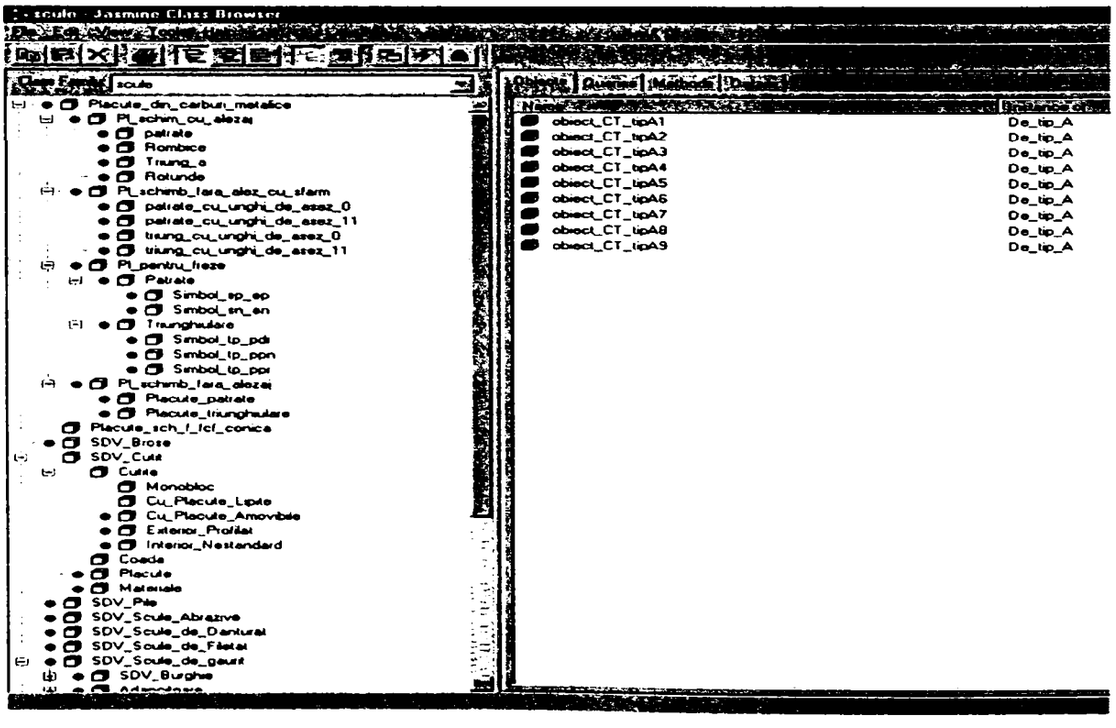


Fig 7.3 Familii de clase utilizator

de clase: familii de clase proprii mediului (System CF şi Media CF) şi familii de clase definite de către utilizator(User CF). În procesul de proiectare a bazei de date trebuie definită încă din prima fază, ierarhia de clase şi ulterior obiectele care aparțin fiecărei clase. Clasele din baza de date sunt prezentate în figura 7.3.

Manipularea obiectelor se realizează cu ajutorul metodelor sau cu ajutorul unor interogări obiectuale (OQL) proprii mediului soft. În general principalele proprietăți teoretice ale unui mediu obiectual:

- Încapsularea;
- Polimorfismul;
- Moştenirea;
- Versionarea;
- Persistența.

sunt respectate în mediul Jasmine. În continuare în acest capitol este prezentat modul în care au fost implementate clasele şi obiectele claselor "Freze cilindrice" şi "Freze cilindro frontale"

7.2 Implementarea frezelor cilindrice

Clasa "Freze cilindrice" face parte din clasa "Freze" care este o subclasă a

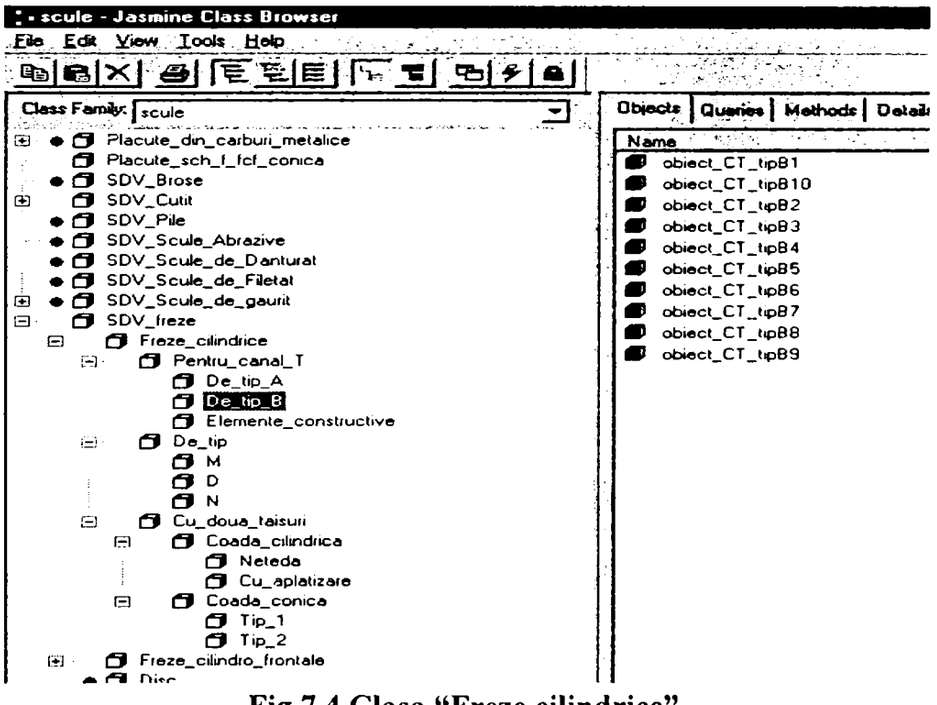


Fig 7.4 Clasa "Freze cilindrice"

clasei "Scule" (figura 7.4)

Clasa "Freze cilindrice " conține mai multe subclase: "De tip", "Cu două tăişuri", "Cu coada conică", care la rândul lor conțin alte subclase:"M","D","N", "Netede", "Cu_aplatizare", "Tip_1", "Tip_2". Clasele de nivel superior nu conțn obiecte. Acestea sunt conținute în clasele de nivelul cel mai de jos. Fiecare clasă sau obiect poate avea ataşată una sau mai multe metode sau interogări cu ajutorul căreia sunt manipulate clasele, respectiv obiectele. Clasa "Freze

cilindrice" este considerată clasă derivată din clasa SDV_freze. Aceasta este o clasă derivabilă. Clasa "Pentru_canale_T" este o clasă derivată prin moştenire din clasa de bază "Freze Ciindrice". Clasele "De_tip_A", "De_tip_B", "Elemente constructive" sunt clase derivate prin moştenire din clasa de bază "Pentru_canal_T"

Clasa "Freze cilindrice" conține o serie de proprietăți care se vor transmite ulterior celorlalte subclase şi obiecte ale subclaselor. Proprietățile clasei sunt prezentate în figura 7.5

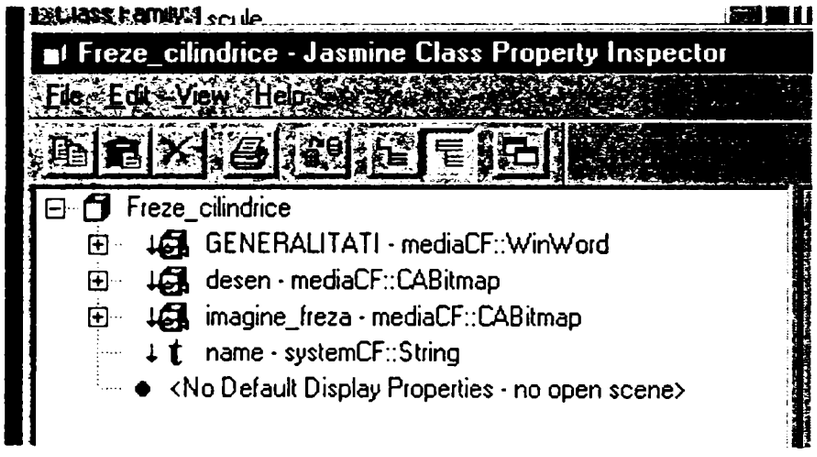


Fig 7.5 Proprietăți ale clasei "Freze cilindrice"

Proprietatea "Generalități" este un document Word în care se găsec informații despre un anumit obiect al clasei. Proprietatea "desen" conține desenul corespunzător obiectului. Proprietatea "imagine" prezintă modul în care arată fizic obiectul (scula) clasei. Acesta poate fi o imagine foto, o imagine obținută în urma proiectării sculei sau un film în care este prezentată scula din toate unghiurile şi prin secțiune. Aceste proprietăți le vor moşteni toate obiectele clasei.

Schema externă a bazei de date obiectuale "Freza cilindrice" este prezentată în figura 7.6.

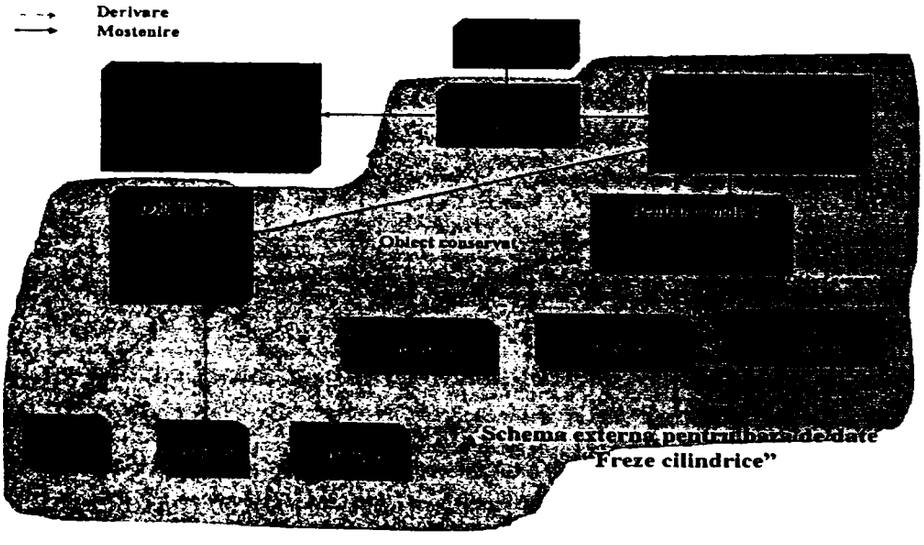


Fig 7.6 Schema externă "Freze cilindrice"

7.2.1 Implementarea Frezelor pentru canale T

Clasa "Freze pentru canale T conține subclasele: "De tip_A", "De tip_B" și "Elemente constructive". Aceste clase precum și obiectele clasei De tip_A sunt realizate în Studio Jasmine și prezentate în figura 7.7.

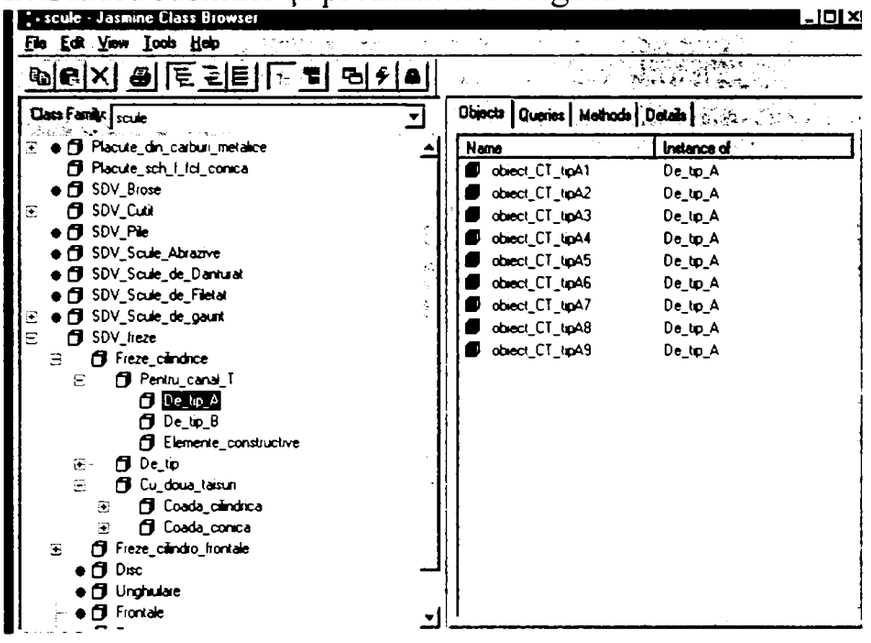


Fig 7.7 Clasa "Freze pentru canale T De tip_A"

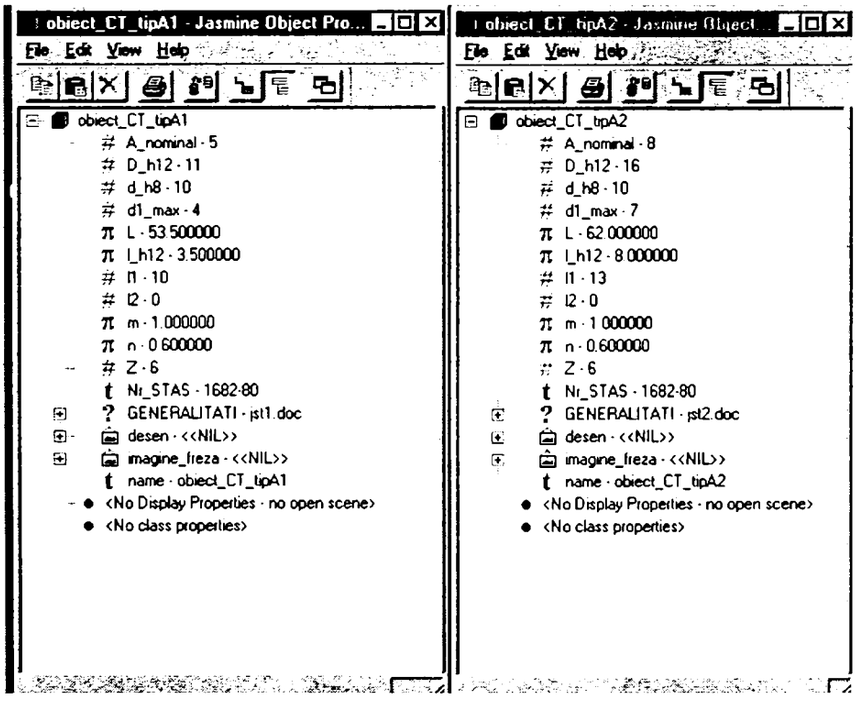


Fig 7.8 Proprietăți ale obiectelor clasei "Freze pentru canale T De tip_A"

Obiectele claselor Tip_A şi Tip_B sunt prezentate în figurile 7.8 şi 7.9. Fiecare obiect are proprietăţi moştenite şi proprietăţi specifice. Proprietăţile specifice pot corespunde atributelor din cadrul unei relaţii. Acelaşi lucru este valabil şi pentru proprietăţile moştenite.

Fiecare proprietate corespunzătoare unui obiect are valori specifice care corespund tuplelor existente în atributele unei relaţii

După cum se observă din cele două figuri, obiectele claselor au o serie de proprietăţi comune moştenite de la superclasa "Freze canale T" precum şi o serie de proprietăţi specifice. În mediul pur obiectual, pot fi definite proprietăţi ale

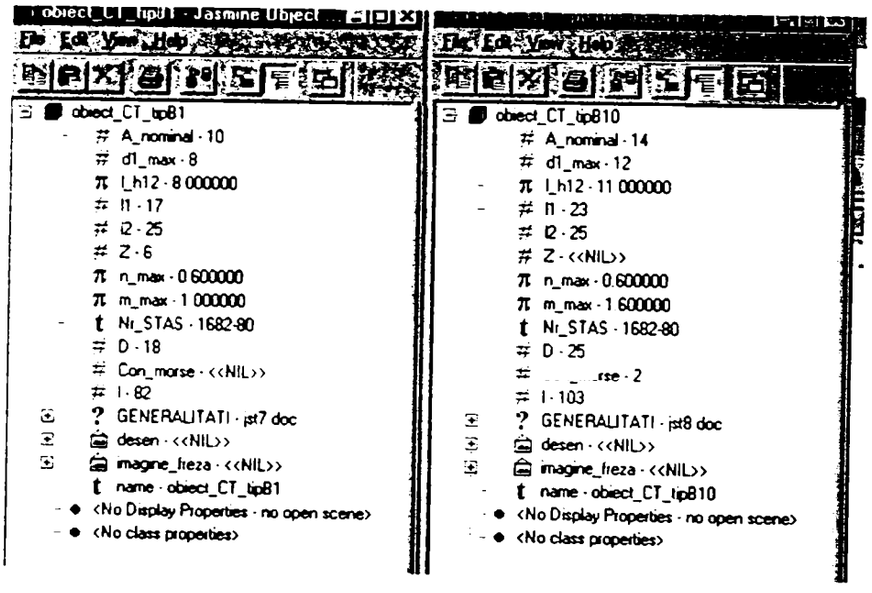


Fig 7.9 Proprietăţi ale obiectelor clasei "Freze pentru canale T De tip_B"

obiectului, care sunt mărimi scalare: întregi, reale, şiruri de caractere etc.

Codul sursă pentru implementarea schemei obiectuale referitoare la "frezele pentru canale T", este scris în limbajul ODQL propriu mediului Jasmine.

Se crează fişierul de memorare al datelor(store) cu numele scule şi familia de clase sculeCF care va conţine toate clasele:

```
Create store –numberofpages 2000 –pagesize 8 scule %jas_system%\doctorat\baza_de_date_scule\
CreateCF –sculeCF scule
```

Aceste comenzi pot fi folosite în fişiere de comenzi separate. În continuare va fi prezentată definirea claselor folosind comenzi ODQL limbajul propriu mediului Jasmine

```
DefaultCF sculeCF

Defineclass scule::sculecomposite
        super: sculeCF::composite
        description: "Clasa abstracta de bază";

Defineclass scule::placute_din_carburi_metalice
        super: scule::sculecomposite
        description: "Clasa container pentru
placute din carburi metalice";

Defineclass scule::placute_sch_f_fcf_conica
        super: scule::sculecomposite
description: "Clasa container pentru placute";

Defineclass scule::SDV_brose
        super: scule::sculecomposite
        description: "Clasa container pentru
brose";

Defineclass scule::SDV_Cutit
        super: scule
```

```
        description: "Clasa container pentru
cutite";

Defineclass scule::SDV_Pile
        super: scule
        description: "Clasa container pentru pile";

Defineclass scule::SDV_Scule_Abrazive
super: scule
description: "Clasa container pentru scule
abrazive";

Defineclass scule::SDV_scule_de_danturat
        super: scule
description: "Clasa container pentru scule de
danturat";

Defineclass scule::SDV_scule_de_gaurit
        super: scule
description: "Clasa container pentru scule de
gaurit";

Defineclass scule::SDV_freze
        super: scule
        description: "Clasa container pentru
freze";

Defineclass sdv_freze::freze_cilindrice
        Super: SDV_freze
        Description:"Clasa freze cilindrice"
{
maxinstance: 4;
instance:
WinWord         GENERALITATI;
CABitmap        desen;
CABitmap        imagine_freza;
string    nume;
};

Defineclass freze_cilindrice::Pentru_canal_T
        Super: freze_cilindrice
        Description:"Freze pentru canale T"
{
MaxinstanceSize: 4;
Instance:
        Integer         A_nominal;
        Integer         d1_max;
        Integer         Z;
        Decimal[5,1]    l1;
        Decimal[5,1]    l2;
        Decimal[9,6]    l_h_12;
        String          Nr_STAS;
        Strng           nume_obiect;
};

Defineclass Pentru_canal_T::Tip_A
        Super: Pentru_canal_T
        Description:"Freze pentru canale T tip A"
{
MaxinstanceSize: 4;
        Instance:
        Integer         d_h8;
        Integer         d_h_12;
        Integer         m;
        Integer         n;
        Decimal[9,6]    L;
};

Defineclass Pentru_canal_T::Tip_B
        Super: Pentru_canal_T
        Description:"Freze pentru canale T tip B"
{
MaxinstanceSize: 4;
        Instance:
        Integer         D;
        Integer         l;
        Integer         Con_morse;
        Decimal[8,6]    n_max;
        Decimal[8,6]    m_max;
};

Defineclass Pentru_canal_T
        Super: Pentru_canal_T
        Description:"Clasa pentru descrierea
        elementelor constructive"
{
MaxinstanceSize 4;
        Instance:
        WinWord:        document1;
        WinWord         document2;
};

buildclass                  scule::sculecomposite
scule::placute_din_carburi_metalice
scule::placute_sch_f_fcf_conica  scule::SDV_brose
scule::SDV_cutit                  scule::SDV_Pile
scule::SDV_scule_abrazive
scule::SDV_scule_de_danturat
scule::SDV_scule_de_gaurit      Scule::SDV_freze
SDV_freze::freze_cilindrice
freze_cilindrice::pentru_canal_T
pentru canal_T::tip_A pentru_canal_T::tip_B

tip_A obiect_tipA;
obiect_tipA=tip_A.new();
obiect_tipA.A_nominal=5;
obiect_tipA.D_h12=11;
obiect_tipA.d_h8=10;
obiect_tipA.d1_max=4;
obiect_tipA.L=53.500000;
obiect_tipA1_h12=3.500000;
obiect_tipA.l1=10;
obiect_tipA.l2=0;
obiect_tipA.m=1.000000;
obiect_tipA.n=0.600000;
obiect_tipA.Z=6;
obiect_tipA.nr_STAS=1682-80;
obiect_tipA.generalitati=jst1.doc;
obiect_tipA.desen=<nil>;
obiect_tipA.imagine_freza=< nil>;
obiect_tipA.nume_obiect=obiect_CT_tipA1
```

```
tip_B obiect_tipB;
obiect_tipB=tip_B.new();
obiect_tipB.A_nominal=10;
obiect_tipB.D=18;
obiect_tipB.l=82;
obiect_tipB.d1_max=8;
obiect_tipB.L=53.500000;
obiect_tipB.l_h12=8.000000;
obiect_tipB.l1=17;
obiect_tipB.l2=25;
obiect_tipB.n_max=0.600000;
obiect_tipB.m_max=1.000000;
obiect_tipB.Z=6;
obiect_tipB.con_morse=<nil>;
obiect_tipB.nr_STAS=1682-80;
obiect_tipB.generalitati=jst7.doc;
obiect_tipB.desen=<nil>;
obiect_tipB.imagine_freza=< nil>;
obiect_tipB.nume_obiect=obiect_CT_tipB1
```

Interfața de prezentare a clasei "Freze pentru Canale T", în software-ul pur obiectual, Jasmine, este prezentată în figura 7.10.

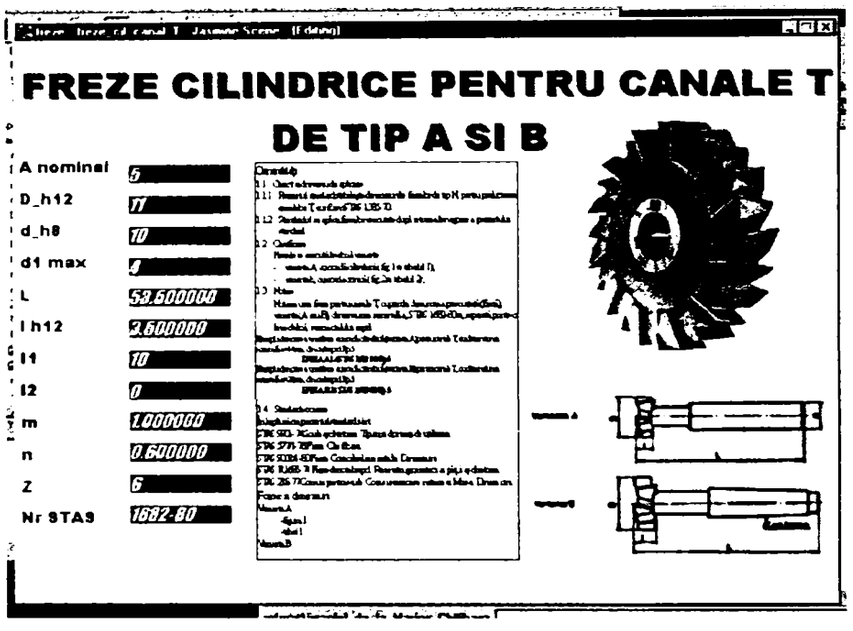


Fig 7.10 Interfața de prezentare a clasei "Freze pentru canale T"

7.2.2 Implementarea Frezelor cilindrice de tip D,M,N

Cealaltă parte importantă a schemei obiectuale, prezentate în figura 7.6 este structura ierarhică de clase corespunzătoare "Frezelor cilindrice de tip D,M,N". Implementarea acestei structuri de clase este asemănătoare cu cea prezentată anterior. Subclasele acestei clase sunt: M,N,D cu obiectele aferente figura 7.11. Fiecare obiect, are o serie de proprietăți ordinale, care sunt prezentate în figura 7.12 pentru subclasa M, figura 7.13 pentru subclasa D şi figura 7.14 pentru subclasa N. Proprietățile, sunt identice cu numele atributelor unei relații, în cazul implementării relaționale. Valorile proprietăților corespund valorilor tuplelor din implementarea relațională aferentă.

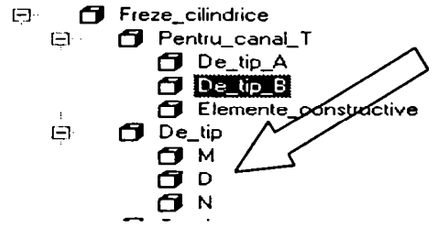


Fig 7.11 Subclasele Clasei "Freze cilindrice"

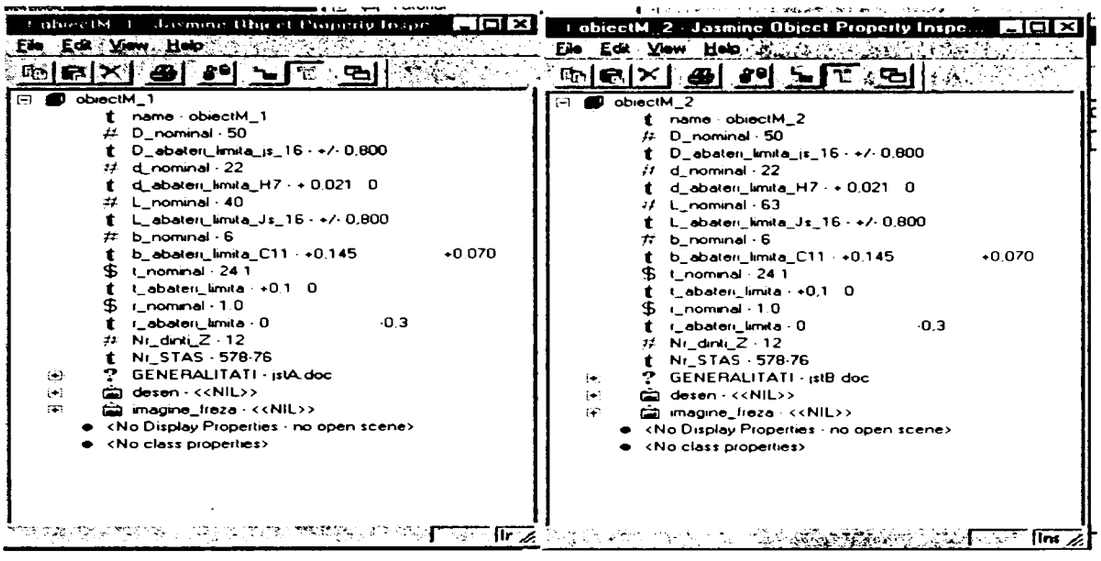


Fig 7.12 Proprietățile obiectelor clasei "Freze cilindrice De tip_M"

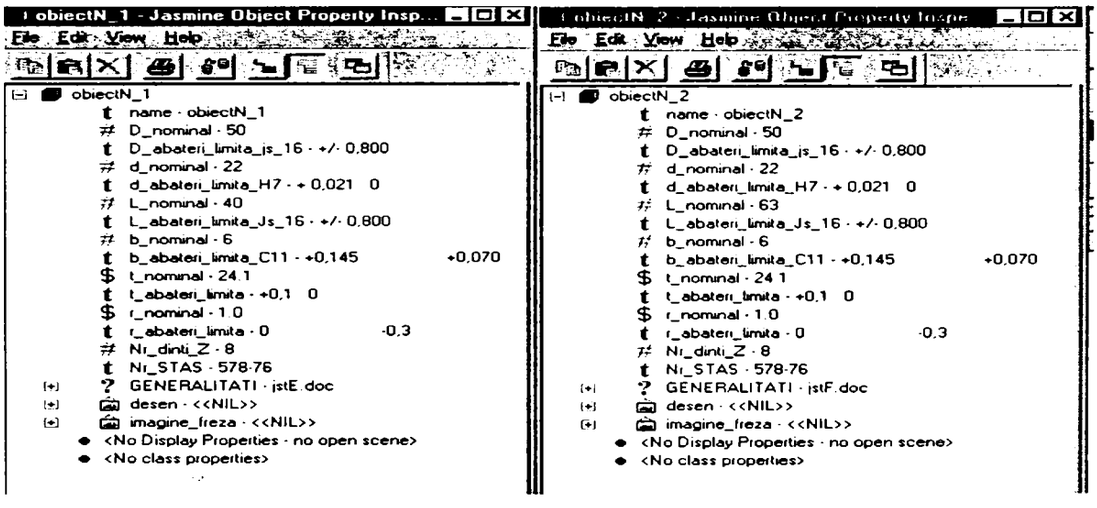


Fig 7.13 Proprietățile obiectelor clasei "Freze cilindrice De tip_N"

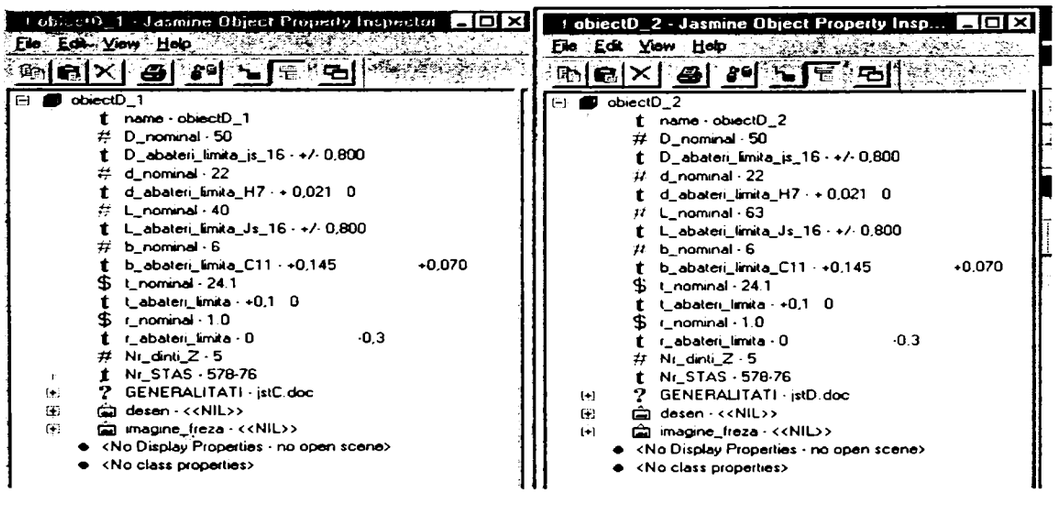


Fig 7.14 Proprietățile obiectelor clasei "Freze cilindrice De tip_D"

Codul sursă scris în ODQL, pentru schema obiectuală şi obiectele prezentate este următorul:

```
Defineclass freze_cilindrice::De_tip
        Super: freze_cilindrice
        Description:"Freze cilindrice de tip M,N
sau D"
{
MaxinstanceSize: 4;
Instance:
        Integer         D_nominal;
        String          D_abateri_limită_js_16;
        Integer         d_nominal;
        String          d_abateri_limită_H7;
        Integer         L_nominal;
        String          L_abateri_limita_Js_16
        Integer         b_nominal;
        String
        b_abateri_limita_C11_16
        Decimal[4.1]    t_nominal;
        String          t_abateri_limita
        Integer         r_nominal;
        String          r_abateri_limita
        Integer         Nr_dinti_Z
        String          Nr_STAS;
        Strng           nume_obiect;
};

Defineclass De_tip::M
        Super: De_tip
        Description:"Freze cilindrice de tip M"
{
MaxinstanceSize: 4;
};

Defineclass De_tip::N
        Super: De_tip
        Description:"Freze cilindrice de tip N"
{
MaxinstanceSize: 4;
};

Defineclass De_tip::D
        Super: De_tip
        Description:"Freze cilindrice de tip D"
{
MaxinstanceSize: 4;
};
```

buildclass freze_cilindrice::De_tip De_tip::M De_tip::N De_tip::D

```
M obiect_tipM;
obiect_tipM=M.new();
obiect_tipM.D_nominal=50;
obiect_tipM.D_abateri_limită_js_16 = +/-0.800;
obiect_tipM.d_nominal=22;
obiect_tipM.d_abateri_limită_H7= + 0,021  0;
obiect_tipM.L_nominal=40;
obiect_tipM.L_abateri_limita_Js_16=+/-0,800;
obiect_tipM.b_nominal=6;
obiect_tipM.b_abateri_limita_C11_16=+0,145
+0,070;
obiect_tipM.t_nominal=1.0       0;
obiect_tipM.t_abateri_limita=+0,1     0
obiect_tipM.r_nominal=1,0;
obiect_tipM.r_abateri_limita=0
obiect_tipM.Nr_dinti_Z=12
obiect_tipM.Nr_STAS=578-76;
obiect_tipM.nume_obiect=M_1;
obiect_tipM.generalitati=jstA.doc;
obiect_tipM.desen=tipM.jpg;
obiect_tipM.imagine_freza=frezaM.jpg

D obiect_tipD;
obiect_tipD=D.new();
obiect_tipD.D_nominal=50;
obiect_tipD.D_abateri_limită_js_16 = +/-0,800;
obiect_tipD.d_nominal=22;
obiect_tipD.d_abateri_limită_H7= + 0,021  0;
obiect_tipD.L_nominal=40;
obiect_tipD.L_abateri_limita_Js_16=+/-0,800;
obiect_tipD.b_nominal=6;
obiect_tipD.b_abateri_limita_C11_16=+0,145
+0,070;
obiect_tipD.t_nominal=24.1       0;
obiect_tipD.t_abateri_limita=+0,1     0
obiect_tipD.r_nominal=1,0;
obiect_tipD.r_abateri_limita=0       -0,3
obiect_tipD.Nr_dinti_Z=5
obiect_tipD.Nr_STAS=578-76;
obiect_tipD.nume_obiect=D_1;
obiect_tipD.generalitati=jstD.doc;
obiect_tipD.desen=<nil>;
obiect_tipD.imagine_freza=<nil>;

N obiect_tipN;
obiect_tipN=N.new();
obiect_tipN.D_nominal=50;
obiect_tipN.D_abateri_limită_js_16 = +/-0,800;
obiect_tipN.d_nominal=22;
obiect_tipN.d_abateri_limită_H7= + 0,021  0;
obiect_tipN.L_nominal=40;
obiect_tipN.L_abateri_limita_Js_16=+/-0,800;
obiect_tipN.b_nominal=6;
obiect_tipN.b_abateri_limita_C11_16=+0,145
+0,070;
obiect_tipN.t_nominal=1.0       0;
obiect_tipN.t_abateri_limita=+0,1     0
obiect_tipN.r_nominal=1,0;
obiect_tipN.r_abateri_limita=0
obiect_tipN.Nr_dinti_Z=12
obiect_tipN.Nr_STAS=578-76;
obiect_tipN.nume_obiect=N_1;
obiect_tipN.generalitati=jstN.doc;
obiect_tipN.desen=tipN.jpg;
obiect_tipN.imagine_freza=frezaN.jpg
```

Implementarea clasei "Freze cilindrice tip M", în software-ul Jasmine este prezentată în figura 7.15

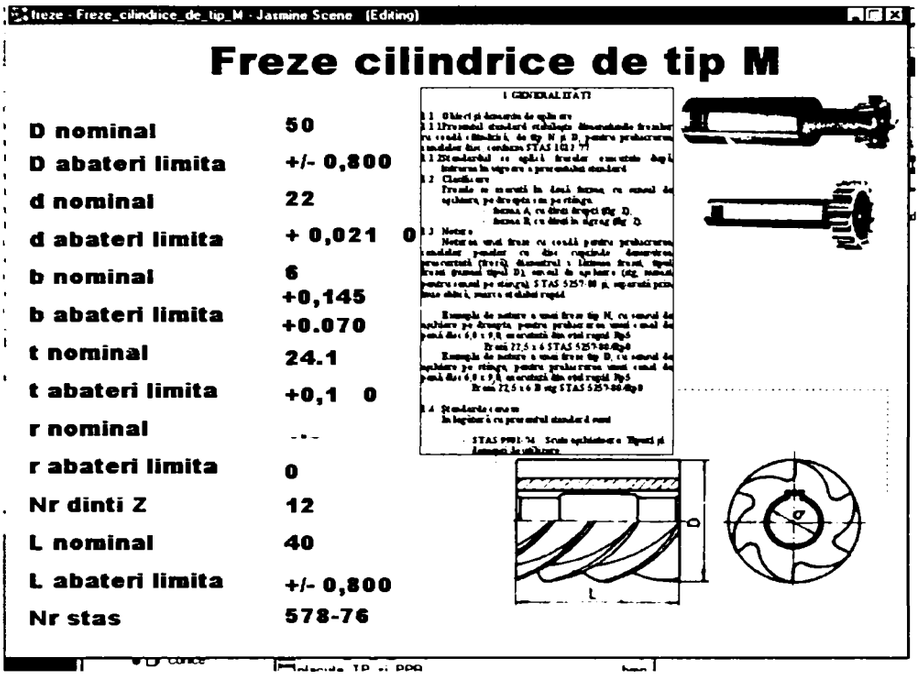


Fig 7.15 Interfața clasei Freze cilindrice De tip_M

Pornind de la schema obiectuală din figura 7.6, s-a prezentat realizarea practică a bazei de date obiectuale. Schema obiectuală considerată a fost generată conform noțiunilor teoretice prezentate în cap. 4 paragrafele 4.2 şi 4.3. Totodată s-a luat în considerare algoritmul de generare a schemelor externe în care toate clasele sunt considerate netransformabile, cap. 5 paragraf. 5.9.1. În general s-a evitat folosirea moştenirii multiple. Clasele inițiale au fost considerate familiile de clase: sculeCF, MediaCF, systemCF, SDV_broşe, SDV_Freze, etc. Clasel sculeCF, MediaCF, SystemCF, Composite, există în dicționarul bazei de date. Clasele care vor fi introduse în schema externă au fost considerate clasele freze_cilindrice, freze_pentru_canale_T, tip_A, tip_B, De_tip, M, N, D. Ele au fost considerate clase netransformabile. Clasele din setul inițial au fost comparate cu mulțimea claselor care vor fi introduse. Perechile de clase prezentate în fig. 7.6, conțin proprietăți comune respectând algoritmul. Totodată clasele prezentate respectă relația de moştenire. Schema obiectuală din figura 6 poate fi considerată o schemă temporală, în sensul că la această schemă poate fi adăugată o mulțime de clase, de exemplu setul de clase "Freze cilindrice cu două tăişuri". Schema obiectuală pentru structura ierarhică de clase "Freze cilindrice" este prezentată în figura 7.16.

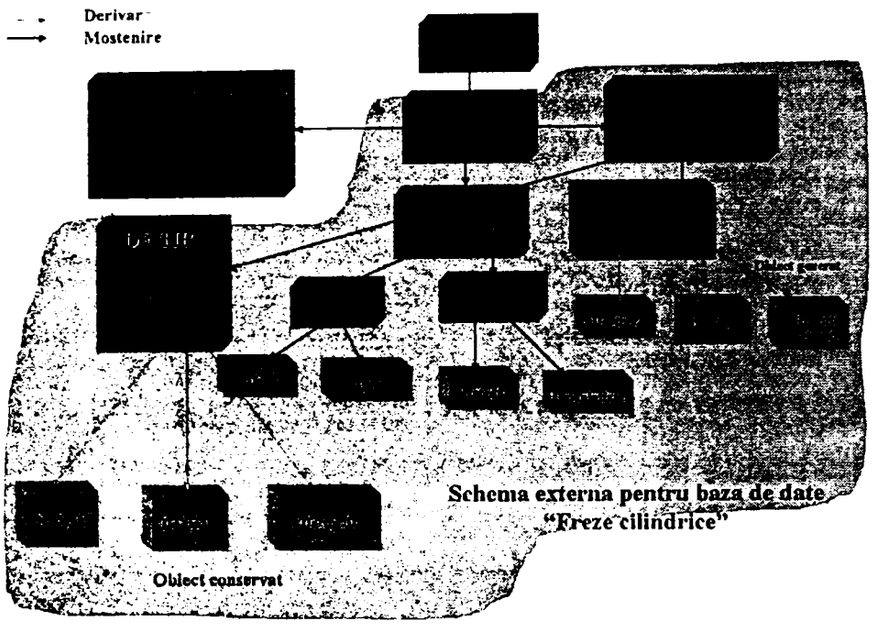


Fig 7.16 Schema externă pentru baza de date obiectuală "Freze cilindrice"

7.2.3 Implementarea Frezelor cilindrice cu două tăişuri cu coadă conică

Proprietățile acestei clase sunt prezentate în figura 7.17. Proprietățile

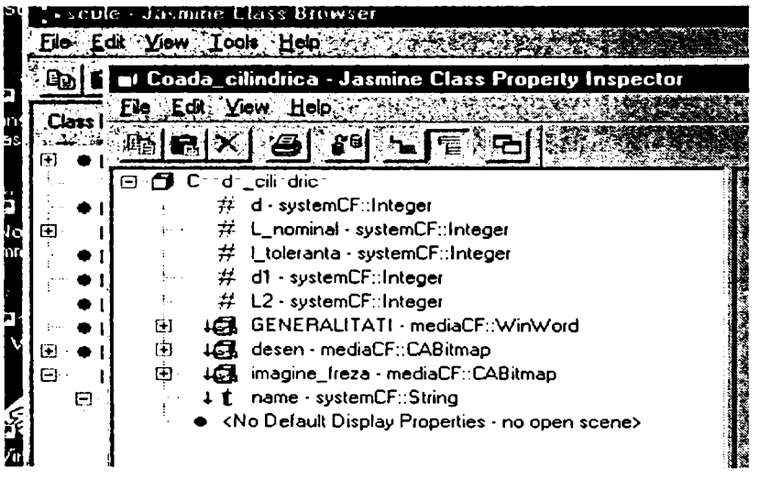


Fig 7.17 Proprietățile ale claselor din schema obiectuală "Freze cilindrice"

"Generalități, desen, imagine" sunt preluate de la proprietățile superclasei de care aparțin. Clasa are o serie de proprietăți specifice care apar la toate obiectele clasei: d, L_nominal, L_toleranță, d1 şi L2.

Obiectele clasei şi proprietățile acestora sunt prezentate în figura 7.18. Acestea au o serie de proprietăți specifice, funcție de tipul cozii. Proprietățile, respectiv valorile lor sunt identice cu atributele, respectiv tuplele, din relațiile bazei de date, corespunzătoare implementării într-un mediu relațional.

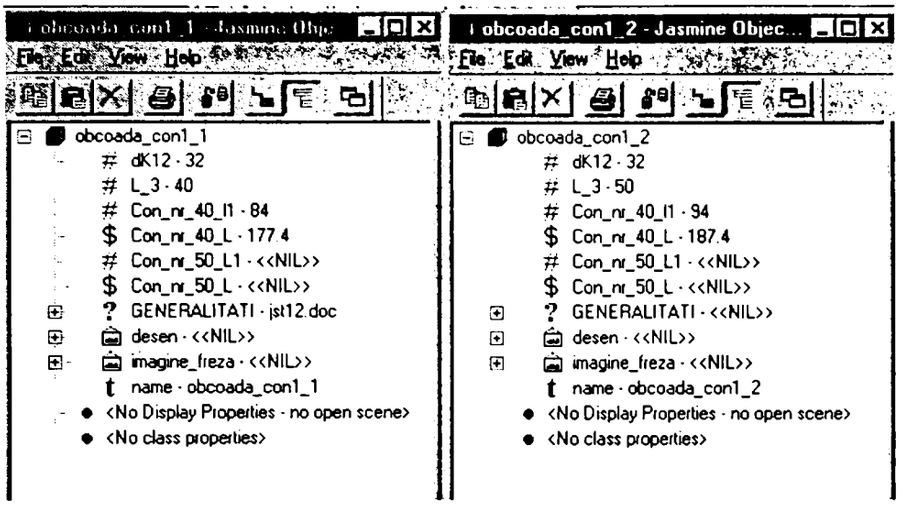


Fig 7.18 Proprietăţile obiectelor clasei "Coadă conică"

Codul sursă al claselor şi al obiectelor este:

```
Defineclass freze_cilindrice::Cu_doua_taisuri
        Super: freze_cilindrice
        Description:"Freze cilindrice cu doua taisuri"
{
MaxinstanceSize: 4;
Instance:
String   name;
};

Defineclass cu_doua_taisuri::coada_cilindrica
        Super: cu_doua_taisuri
        Description:"Freze cilindrice cu coada cilindrica"
{
MaxinstanceSize: 4;
Instance:
Integer          d;
Decimal[5,1]     Con_nr_50_L;
};
buildclassfreze_cilindrice::cu_doua_taisuri
cu_doua_taisuri::coada_conica
cu_doua_taisuri::coada_cilindrica
 coada_cilindrica obcoada_cil;
obcoada_cil=coada_cilindrica.new();
obcoada_cil.d =10;
obcoada_cil.L_nominal=6;
obcoada_cil.L_toleranta=1;
obcoada_cil.d1=11;
obcoada_cil.L2=9
obcoada_con.dK12=32;
obcoada_con.L_3=40;
Integer          L_nominal;
Integer          l_toleranta;
Integer          d1;
Integer          L2;
};

Defineclass cu_doua_taisuri::coada_conica
        Super: cu_doua_taisuri
        Description:"Freze cilindrice cu coada conica"
{
MaxinstanceSize: 4;
Instance:
Integer          dK12;
Integer          L_3;
Integer          Con_nr_40_l1;
Decimal[5,1]     Con_nr_40_L;
Integer          Con_nr_50_l1;
obcoada_cil.nume_obiect=obcoada_cil1;
obcoada_cil.generalitati=<nil>;
obcoada_cil.desen=<nil>;
obcoada_cil.imagine_freza=<nil>;

coada_conica obcoada_con;
obcoada_con=coada_conica.new();
obcoada_con. con_nr_40_L=177.4;
obcoada_con.con_nr_40_l1=84;
obcoada_con.con_nr_50_l1=<nil>;
obcoada_con. con_nr_40_L=<nil>;
```

Implementarea clasei "Freze cilindrice cu două tăişuri cu plăcuţe elicoidale cu coadă conică Tip1", este prezentată în figura 7.19

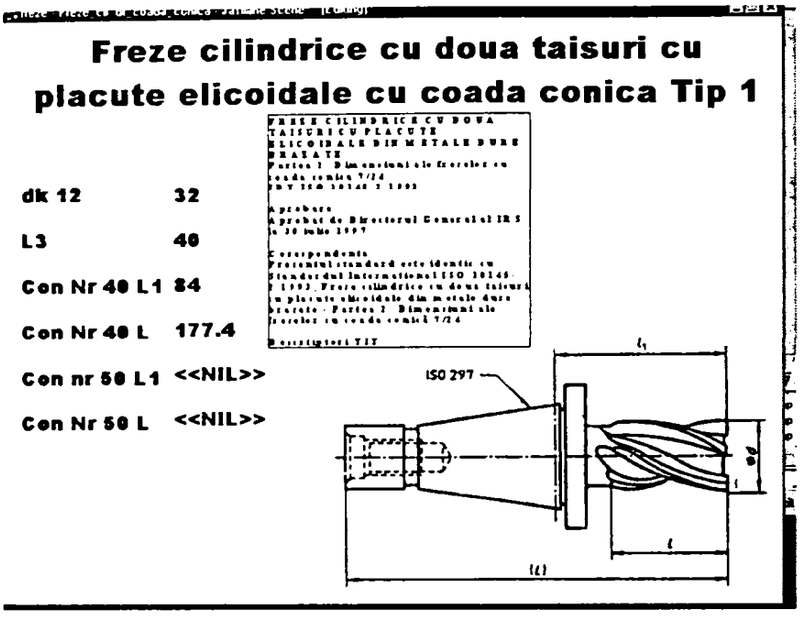


Fig 7.19 Freze cilindrice cu două tăişuri cu plăcuțe elicoidale cu coadă conică Tip1

Modul de implementare a "Frezelor cilindrice cu două tăişuri cu plăcuțe elicoidale din metale dur brazdate, cu coadă cilindrică" este asemănătoare, rezultatul implementării clasei fiind prezentat în figura 7.20.

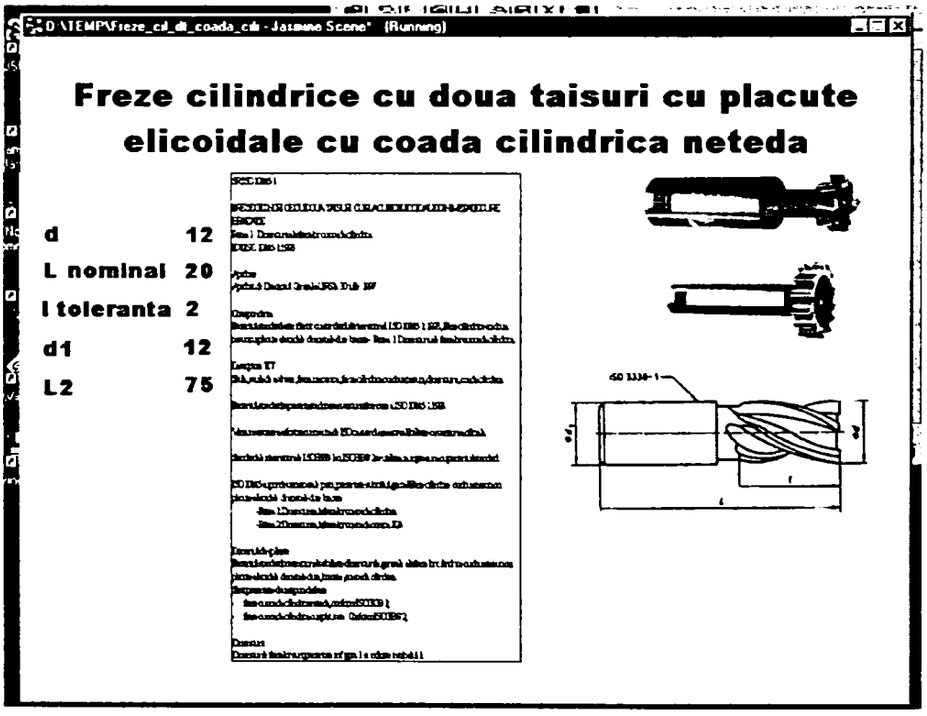


Fig 7.20

7.3 Implementarea frezelor cilindro frontale

Implementarea celorlalte clase este asemănătoare. În figura 7.21, este prezentată schema obiectuală temporală, a setului de clase "Freze cilindro-frontale.

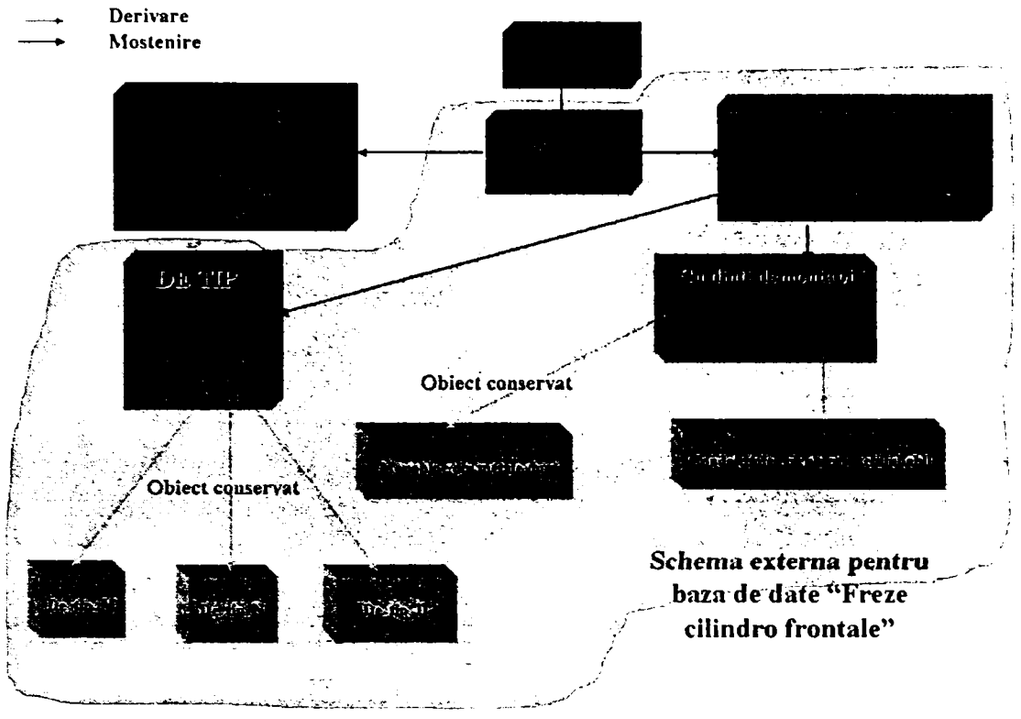


Fig 7.21 Schema obiectuală pentru baza de date "Freze cilindro frontale"

Ea este prezentată, cu ajutorul Class Browser-ului existent în Jasmine Studio. Schemele externe s-au realizat, folosind algoritmul prezentat în cap. 5 paragraf. 5.9.2. De această dată s-a pornit de la conceptul de clasă transformabilă, luându-se în considerare intensia şi mai ales extensia claselor

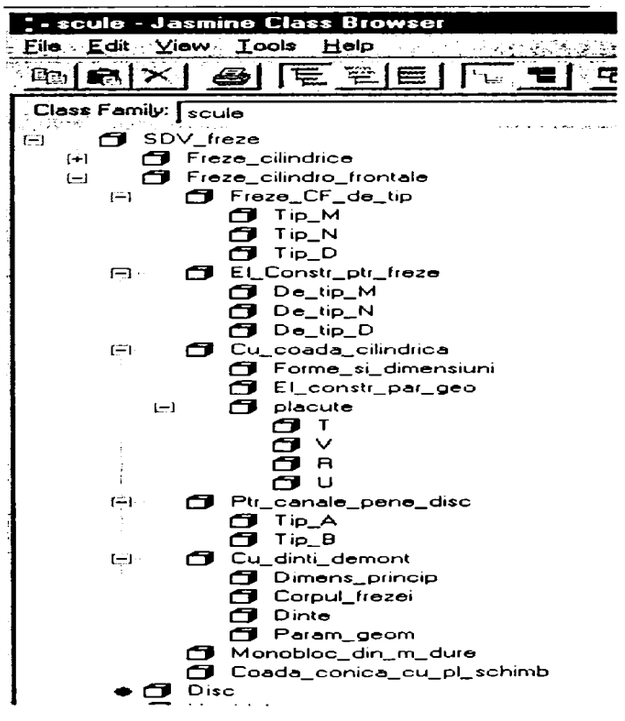


Fig 7.22 Structura arborescentă a setului de clase "Freze_cilindro_frontale" în software-ul Jasmine

Inițial elementele constructive pentru freze au fost atribute ale clasei "Freze_cilindro_frontale". Prin derivare folosind relația de moștenire s-a obținut clasa netransformabilă "El_constr_ptr_freze". Clasa "Freze_cilindro_frontale" a fost considerată clasă transformabilă. În schema temporală ea este considerată în continuare clasă transformabilă. Clasele care se obțin sunt considerate clase derivabile obținute prin relația de moștenire.

7.3.1 Implementarea frezelor cilindro frontale de tip N,M,D

Modul de implementare al obiectelor acestui tip de clasă este asemănător

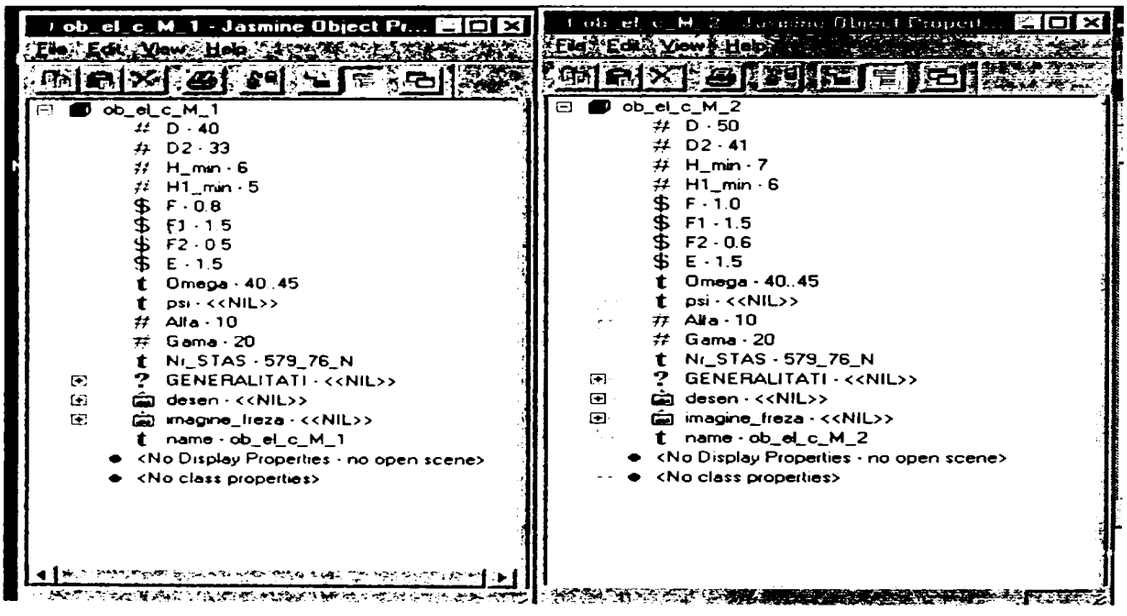


Fig 7.23 Obiecte ale clasei M

cu cele prezentate anterior. Obiectele clasei M sunt prezentate în figura 7.23. Modul de proiectare şi implementare a claselor şi a obiectelor este asemănător

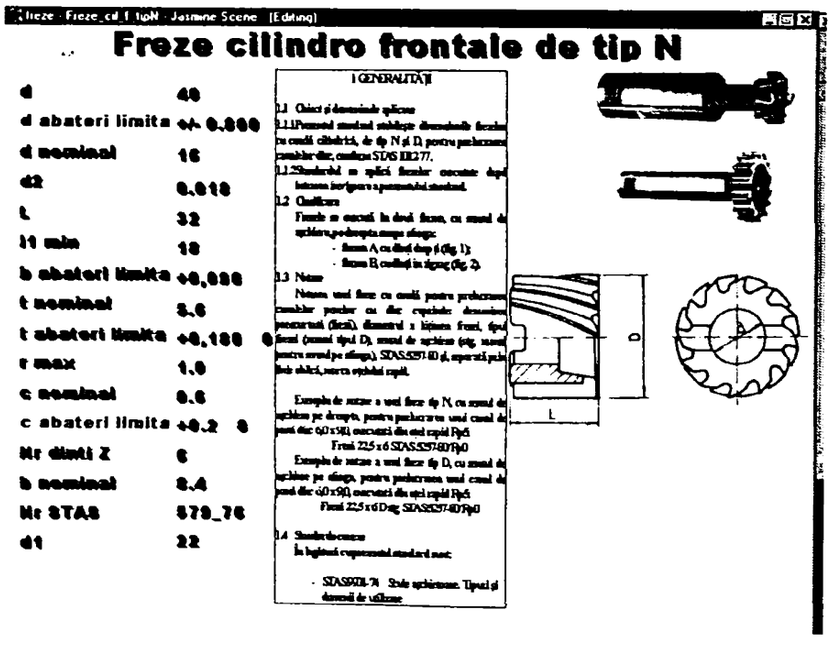


Fig 7.24 Clasa Freze cilindro frontale de tip N

cu cel prezentat la frezele cilindrice.
Implementarea în mediul CA Jasmine, a clasei "Freze cilindro frontale de tip N" este prezentată în figura 7.24

7.3.2 Implementarea frezelor cilindro frontale din metale dure

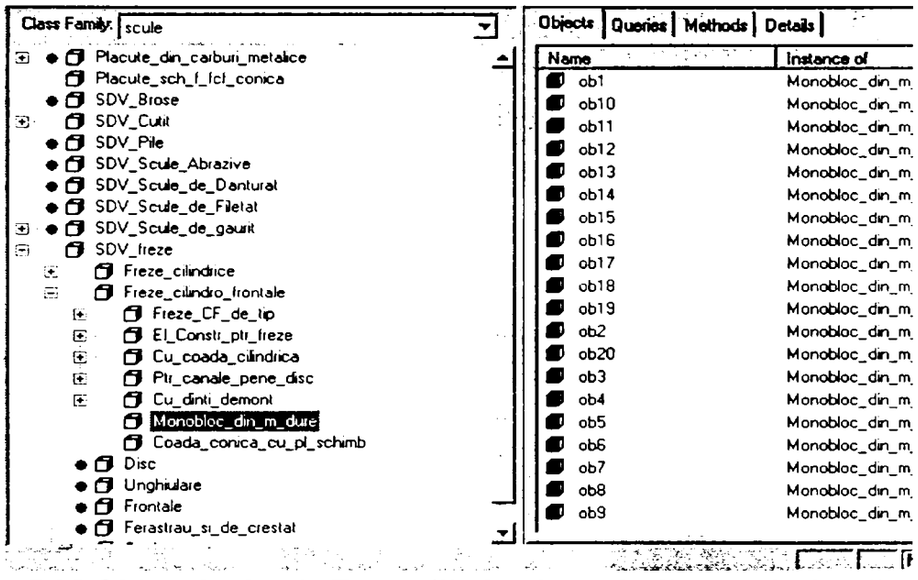


Fig 7.25 Obiectele clasei "Freze cilindro frontale din metale dure"

Implementarea clasei şi a obiectelor specifice clasei este prezentată în figura 7.25
Proprietățile obiectelor clasei sunt prezentate în figura 7.26.

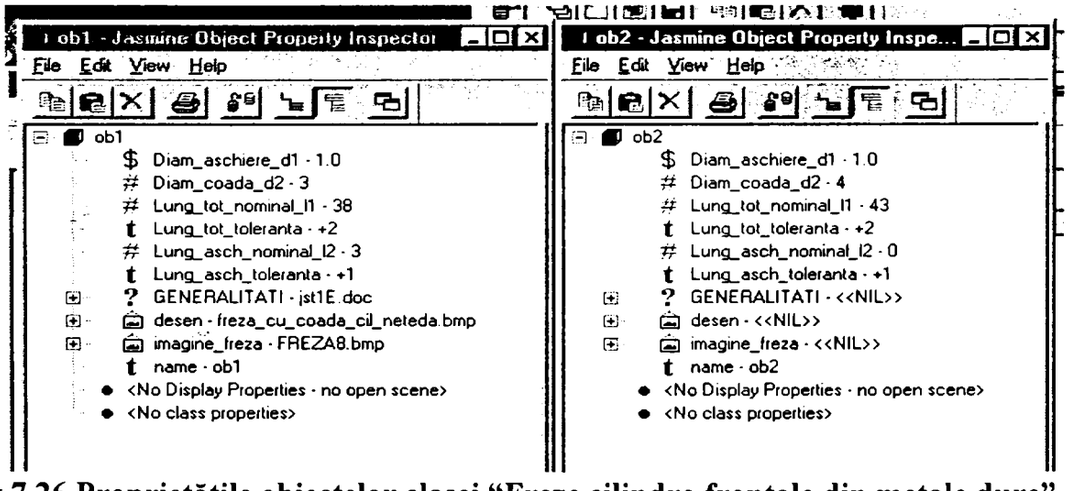


Fig 7.26 Proprietățile obiectelor clasei "Freze cilindro frontale din metale dure"

Modul în care sunt folosite obiectele acestei clase într-o aplicație CA Jasmine este prezentat în fig 7.27

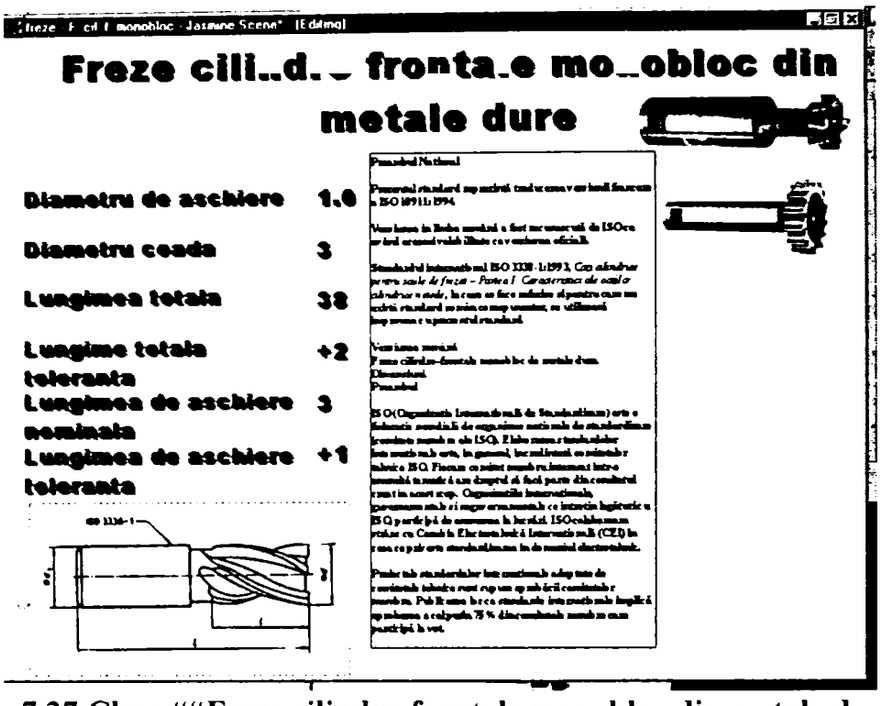


Fig 7.27 Clasa ""Freze cilindro frontale monobloc din metale dure"

7.3.3 Implementarea frezelor cilindro frontale cu coadă cilindrică cu plăcuțe lipite din carburi metalice

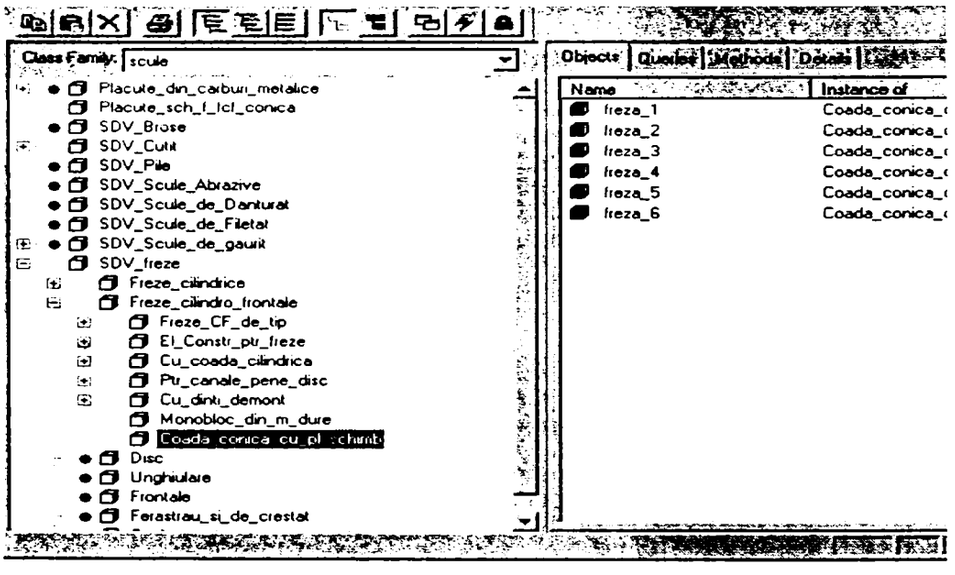


Fig 7.28 Obiectele clasei "Freze cilindro frontale cu coadă cilindrică cu plăcuțe lipite din carburi metalice"

Obiectele clasei sunt prezentate în figura 7.28. Proprietățile obiectelor sunt prezentate în fig 7.29.

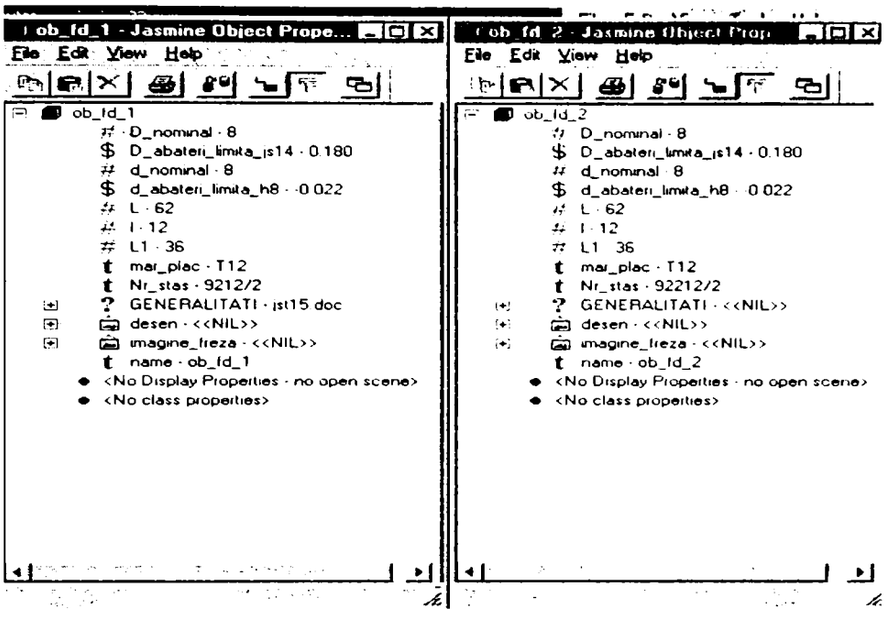


Fig 7.29 Proprietățile obiectelor "Freze cilindro frontale cu coadă cilindrică cu plăcuțe lipite din carburi metalice"

Modul în care sunt folosite obiectele acestei clase, într-o aplicație CA

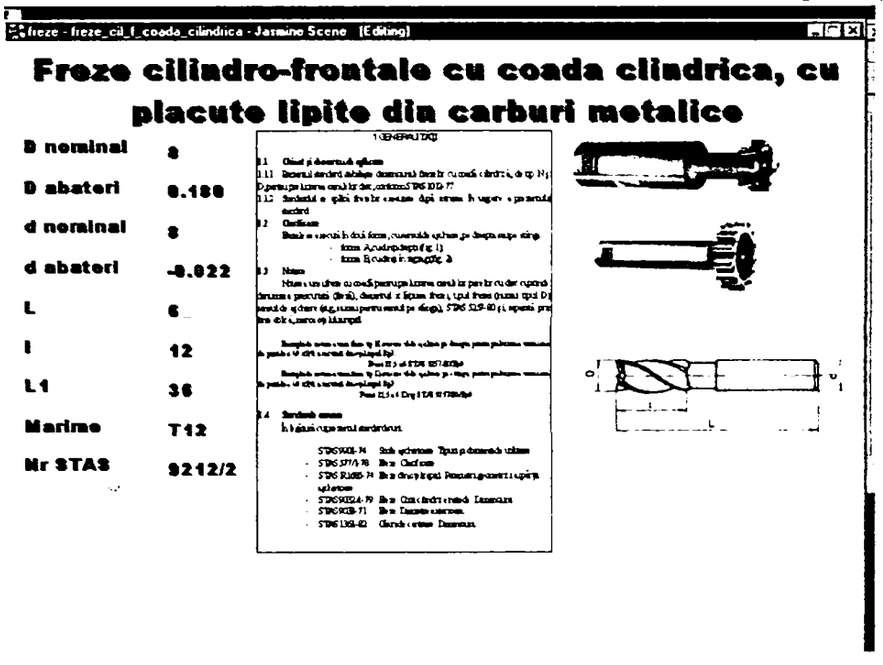


Fig 7.30 Clasa "Freze cilindro frontale cu coadă cilindrică cu plăcuțe lipite din carburi metalice"

Jasmine, este prezentat în fig 7.30

Datorită faptului, că modul de realizare a claselor şi a obiectelor este asemănător cu cele prezentate în cazul clasei "Freze cilindrice", nu s-a mai prezentat codul program aferent.

7.4 Proiectarea în detaliu a bazei de date pentru sistemul de fabricație al sculelor

Realizarea celorlalte clase, a obiectelor şi a modului de folosire într-o aplicație specifică, este asemănător cu cel al claselor prezentate la începutul capitolului.

Manipularea obiectelor din cadrul claselor se realizează cu ajutorul metodelor ce pot fi ataşate claselor sau obiectelor. O altă modalitate de a transmite mesaje între obiecte, se poate realiza cu ajutorul interogărilor obiectuale (query) sau cu ajutorul unor mecanisme proprii mediului jasmine Studio fig 7.31.

Fig 7.31 Realizarea interogărilor obiectuale cu unelte specifice software-ului Jasmine

Se alege obiectul dorit şi se definesc mesajele pe care acesta le va transmite (Message) precum şi acțiunile care au loc în urma acestora (Action).

În cazul în care se doreşte scrierea unei interogări care afişează frezele pentru canale T care au diametrul mai mare decât 32 se va scrie următorul cod program:

```
Defineclass Pentru_canal_T::Tip_B
        Super: Pentru_canal_T
        Description:"Freze pentru canale T tip B"
{
MaxinstanceSize: 4;
        Instance:
        Integer         D;
        Integer         l;
        Integer         Con_morse;
        Decimal[8,6]    n_max;
        Decimal[8,6]    m_max;
};
buildclass tip_B;
tip_B f;
bag <tip_B> freza_tip_B;
freza_tip_b=[tip_B.D, tip_B.l, tip_B.n_max,
tip_B.m_max] where tip_B.D>=32;
scan (freza_tip_b, f) {f.D.print(), f.lprint(),
f.n_max.print(), f.m_max.print(); };
```

Rezultatul unei interogări se poate vizualiza cu ajutorul unui obiect "tupla", care conține mai multe valori ataşate unei clase, sau ca rezultat al unei interogări.

```
List <TT [integer D, integer l] > obfreza;
Obfreza = [f.D, f.l] from tip_B f where  f.d>=32;
Obfreza.print();
```

În acest caz interogarea returnează o colecție de tuple, fiecare element conține diametrul şi lungimea cu proprietatea că diametrul este mai mare decât 32. Cu ajutorul interogărilor pot fi definite clase derivate în OODB-ul Jasmine.

În concluzie, modul de implementare într-un mediu obiectual, este diferit de modul de implementare într-un mediu hibrid, realizarea aplicației se apropie din punct de vedere conceptual, foarte mult de teoria pur obiectuală a bazelor de date. S-a prezentat modul în care sunt realizate efectiv clasele de obiecte într-un mediu pur obiectual, precum şi schemele obiectuale aferente.

Modul în care se proiectează baza de date obiectuală, pentru sistemul de fabricaţie al sculelor, poate fi rezumat în figura 7.32

Pe baza modului în care s-a realizat proiectarea bazei de date pentru freze

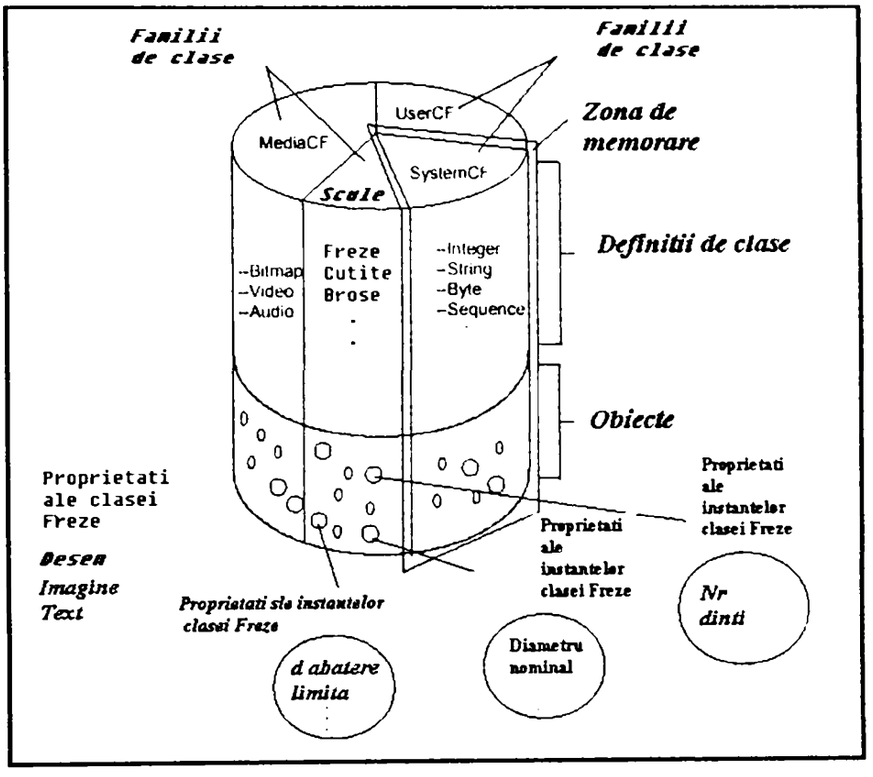


Fig 7.32 Baza de date obiectuală pentru sistemul de fabricaţie al sculelor

cilindrice şi freze cilindro frontale se va realiza proiectarea celorlalte ramuri importante ale bazei de date.

Ţinând cont de tendinţa actuală pe piaţa bazelor de date, în care SGBDOR-le, adică bazele de date relaţional obiectuale sunt cele mai des utilizate, ocupând primul loc în ceea ce priveşte vânzările şi răspândirea globală, s-a ales soluţia folosirii unui astfel de mediu soft pentru realizarea unei implementări pur obiectuale pe un mediu relaţional a aplicaţiei de gestionare a SFS-ului.

8

Implementarea obiectuală a bazei de date pentru Sistemul de Fabricatie al Sculelor, într-un mediu relaţional obiectual

Cu ani în urmă OODBMS-urile erau considerate tehnologia viitorului, adică bazele de date care se pretează cel mai bine la tehnologia Internet. Se considera ca tehnologia OODBMS va deveni principala tehnologie în domeniul bazelor de date, ea înlocuind sistemele relaţionale care nu erau prevazute cu tipuri de date multimedia, foarte des întâlnite pe Internet. Mai mult decăt atât se prevedea că, creşterea numărului de reţele de tip Intranet, va duce la scăderea numărului de reţele în care este implementată tehnologia client-server, care se bazează pe modelul relaţional al bazelor de date. Astăzi s-a dovedit că aceste predicţii au fost în general eronate. Bazele de date relaţionale continuă să fie cele mai uzuale baze de date. Au apărut ORDBMS-urile care sunt sisteme de baze de date relaţional obiectuale care au adăugat caracteristici obiectuale la bazele de date relaţionale. Ele au câştigat în popularitate, şi se aşteaptă ca până în 2003 fiecare RDBMS să aibă implementate caracteristici obiectuale. Vânzările de baze de date relaţionale au fost considerabil mai mari decât OODBMS-urile. Cu toate acestea, Rick Cattel inginer la Sun Microsystems, consideră că decesul acestui tip de baze de date, este foarte departe. Ele sunt folosite cu succes în domeniul CAD şi în domeniul telecomunicaţiilor. Michael Stonebraker, unul dintre pionierii ORDBMS-urilor de la Informix, spune că OODBMS-urile ocupă o mică parte din vânzările actuale, dar bazele de date obiectual relaţionale, vor reprezenta principalul segment de vânzări în următorii 5 ani.

Conform estimărilor IDC pe piaţa mondială vânzările de baze de date relaţionale şi obiectual relaţionale se ridică la 11,1 miliarde de dolari în 1999, pe când vânzarile bazelor de date obiectuale se ridică la 211 milioane dolari. În 2001 vânzările au fost de 15,6 miliarde dolari pentru RDBMS-uri şi ORDBMS-uri şi 265 milioane dolari pentru OODBMS-uri. Pentru 2004 se preconizează o rată de creştre de 18,2% pentru ORDBMS-uri şi 12,5% pentru OODBMS-uri.

OODBMS-urile nu sunt principalul segment în ceea ce priveşte vânzările dar faptul că suportă obiecte complexe, sunt folosite cu precădere în domeniul inteligenţei artificiale şi domeniul CAD/CAM. Pentru aceste domenii folosind OODBMS-uri se reduce codul program folosit, timpul de realizare al aplicaţiei

precum şi timpul de întreţinere al aplicaţiei. În [Leawi 00], Akmal Chaudri profesor la City University of London, afirmă că folosind OODBMS-uri, la proiectarea lui Boeing 747, legăturile dintre părţile componente ale avionului sunt controlate de către baza de date. Dacă se folosesc baze de date relaţionale trebuie descompus avionul în tabele şi apoi legate tabelele între ele, atunci când se reconstruieşte avionul. În cazul OODBMS-urilor, volumul mare de tranzacţii este mai bine contorizat de către utilizatori iar preţul acestor este mai mic. Cu toate că au unele avantaje evidente, bazele de date pur obiectuale nu deţin pe piaţa mondială principalul segment, aşa cum s-a arătat anterior. O arhitectură a unei baze de date obiectuale distribuite, în particular Objectivity arată ca în figura 8.1.

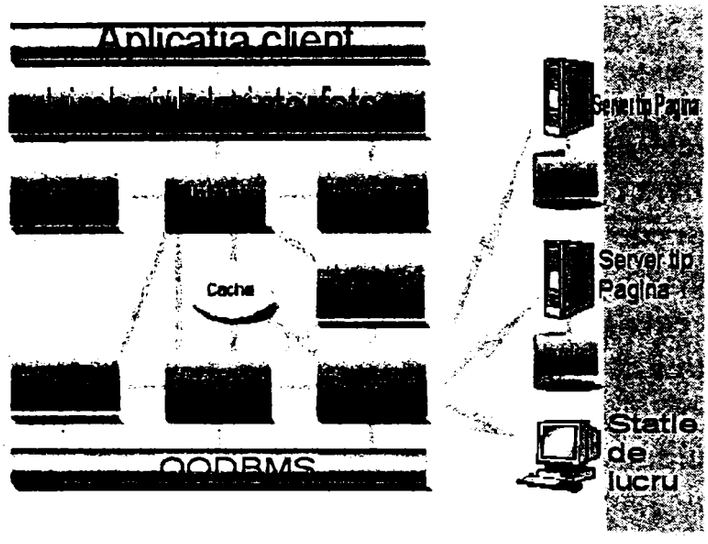


Fig 8.1 Arhitectura Objectivity

Spre deosebire de baza de date relaţională, baza de date Objectivity, necesită un manager de obiecte care manipulează obiectele, le dă nume, le crează funcţie de cerinţele aplicaţiilor client server. Totodată există un manager de memorare a obiectelor care are rolul de a modifica dimensiunile acestora, folosind clusteri de obiecte. Memorarea datelor în bazele de date relaţionale se face pe baza calculului tabelar, ele lucrând cel mai bine cu date care au aceaşi mărime şi structură.

Companiile folosesc analiza şi proiectarea orientată obiect pentru a-şi modela obiectele, deoarece acest lucru îi ajută în înţelegerea comportamentului produselor şi a structurii acestora. Orientarea obiect conţine diverse mecanisme ca de exemplu moştenirea care ajută la proiectarea rapidă a unor noi produse. Unele concepte proprii OODBMS-urilor au limitat vânzările acestora pe piaţa bazelor de date.

Producătorii de baze de date relaţionale au introdus elemente obiectuale în cadrul acestora, pentru a contracara ameninţarea bazelor de date obiectuale. Astfel ORDBMS-urile suportă programarea orientată obiect tabelele sunt

transformate în clase de obiecte, fiecare obiect al clasei fiind o tuplă [Jepso 02]. Totodată au fost introduse tipuri de date care suportă obiectele multimedia.

OODBMS-urile pot memora în interiorul lor set-uri de date astfel că ele pot rula mai rapid decât bazele de date relaţionale. Bazele de date orientate obiect pot stoca automat datele în memoria aplicaţiilor client, eliminând în acest fel o serie de cereri suplimentare ale sistemului, crescând astfel în mod semnificativ viteza de lucru a aplicaţiei. Totodată OODBMS-urile folosesc algoritmi de optimizare care folosesc cele mai bune căi pentru rezolvarea interogărilor. La nivel de standardizare, SQL-ul specific bazelor de date relaţionale a fost adoptat de către ISO (International Organisation for Standardization) şi de către ANSI (American National Standards Institute). ODMG a realizat standardizările la nivelul bazelor de date orientate obiect [Catte 97], [Catte 00]. A fost creat standardul OQL pentru interogările obiectuale care a fost implementat de puţini producători de baze de date OODBMS vor fi folosite în continuare în aplicaţiile în care îşi justifică prezenţa rămânând în topul tehnologiilor referitoare la bazele de date. Tocmai datorită faptului că bazele de date obiectuale, sunt importante, producătorii de baze de date relaţionale vor sisteme obiectual relaţionale, care vor lua locul celor relaţionale [Staja 98].

Pornind de la aceste consideraţii s-a ales pentru implementarea sistemului de fabricaţie al sculelor un sistem *obiectual relaţional* Visual Fox.

8.1 Definirea obiectelor de date într-un mediu relaţional obiectual

Definirea obiectelor de date este un proces prin care se încearcă păstrarea caracteristicilor relaţionale ale bazei de date într-un mediu obiectual. În acest scop se păstrează relaţiile care sunt definite ca şi clase de un tip predefinit (*cursor*) figura 8.2, baza de de date a mediului relaţional este definită ca şi clasă predefinită (*dataenvironment*) figura 8.3. În acestă clasă vor fi ataşate obiecte de tipul claselor relaţiilor din baza de date. Totodată legăturile existente între atributele relaţiilor în cadrul bazei de date, sunt definite ca şi clase de tip predefinit (*relation*). Vederile din cadrul bazei de date sunt definite ca şi clase de tip predefinit(*cursor*).

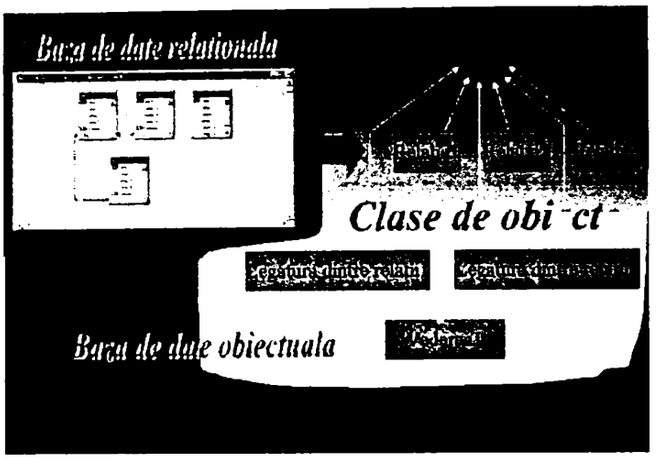


Fig 8.2 Clase de obiecte pentru memorarea datelor

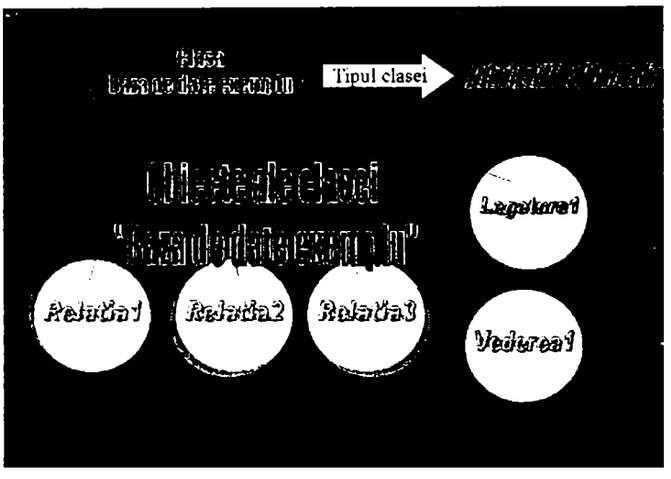


Fig 8.3 Clasa pentru memorarea bazei de date relaționale

Clasele predefinite posedă un set propriu de proprietăți şi metode. Proprietățile şi metodele clasei *cursor* sunt prezentate în figura 8.4

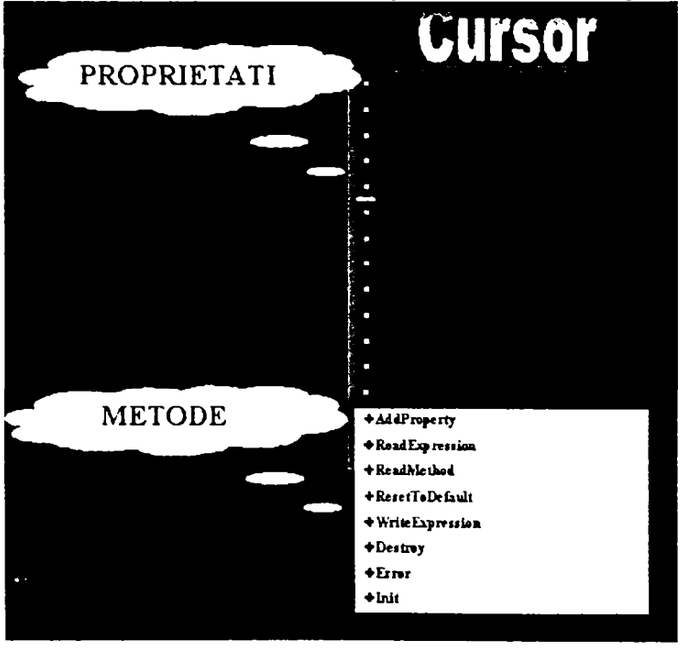


Fig 8.4 Proprietățile şi metodele clasei cursor

Proprietățile și metodele clasei dataenvironment sunt prezentate în figura 8.5.

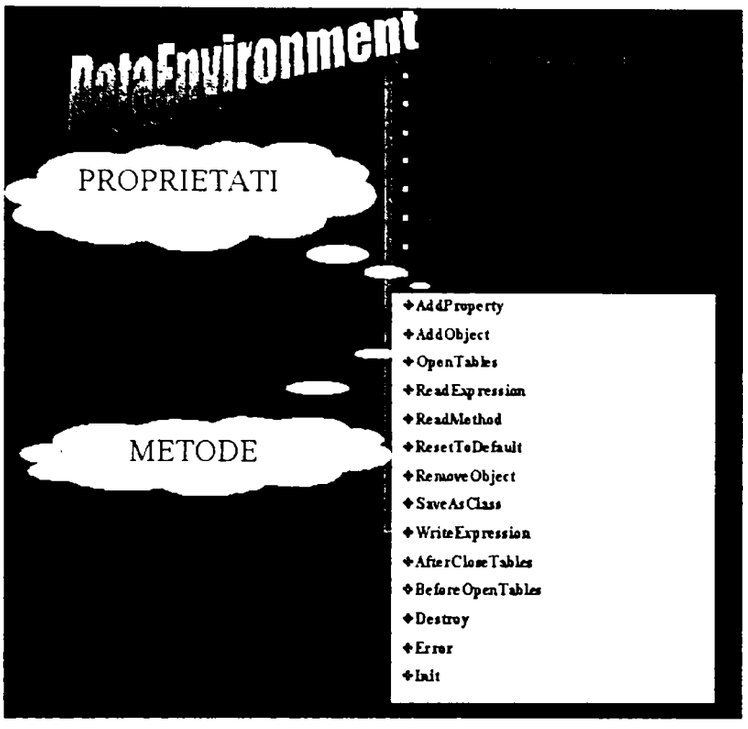


Fig 8.5 Proprietățile și metodele clasei dataenvironment

Aceste două tipuri de clase vor sta la baza realizării bazei de date obiectuale pentru sistemul de fabricație al sculelor. Toate relațiile și vederile vor fi clase de tip *cursor*, iar în clasa *dataenvironment* vor exista obiecte a acestor tipuri de clase. Conceptele: clasă derivată, clasă derivată parțial, scheme externe etc vor fi aplicate pentru realizarea bazei de date obiectuale. Un mod de abordare particular îl constitue realizarea bazelor de date obiectuale corespunzătoare fiecărui tip de sculă cu ajutorul unor clase specifice care au drept scop, generarea codului sursă pentru fiecare bază de date obiectuală

Pentru definirea schemei obiectuale a bazelor de date s-a lucrat cu produsul Microsoft Visual Modeler. Din acest punct de vedere, clasele cursor și dataenvironment sunt prezentate în figura 8.6. Visual Modeler este un mediu pentru modelarea grafică a obiectelor. Una dintre însușirile de bază a acestui mediu soft este aceea că permite realizarea schemelor obiectuale (diagramelor de clasă) a bazelor de date relativ simplu și ușor. Totodată permite crearea automată de cod program în Visual Basic sau C++ sau generarea automată a schemelor obiectuale din fișierele de tip clasă a mediului VisualFox.

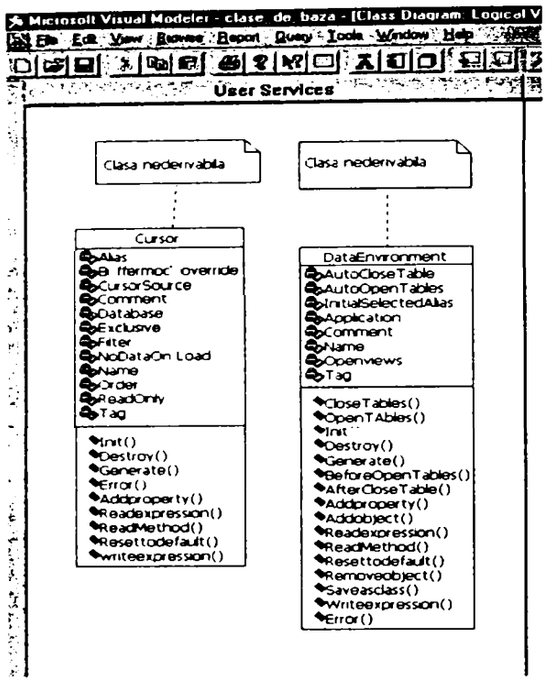


Fig 8.6 Schema obiectuală a claselor Cursor şi Dataenvironment

8.2 Definirea claselor pentru generarea obiectelor de date

Crearea bazei de date orientate obiect în mediul relațional obiectual are la bază, definirea claselor pentru mediile de stocare a datelor. Clasele pentru stocarea datelor sunt generate prin cod program. Ca un element deosebit al acestei teze atât din punct de vedere al concepției cât şi al realizării este faptul că, codul program este generat la rândul său de către clase de obiecte. Elementul de noutate constă în modul de abordare al problemei, adică generarea codului entru clasele ce conțin obiectele de date, prin clase de obiecte.

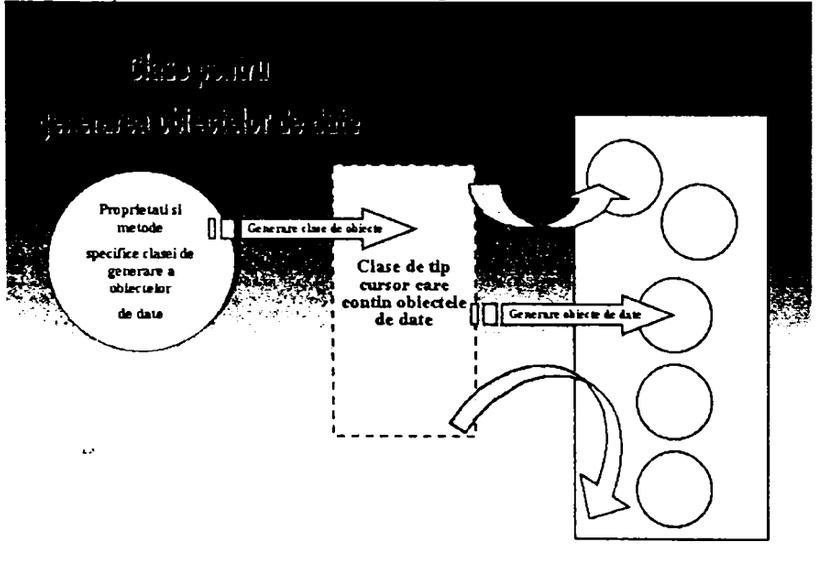


Fig 8.7 Generarea claselor pentru memorarea datelor

Se poate vorbi astfel despre aspectul obiectual general al aplicației, figura 8.7.

Clasele pentru generarea obiectelor de date, modelate în Visual Modeler sunt prezentate în figura 8.8.

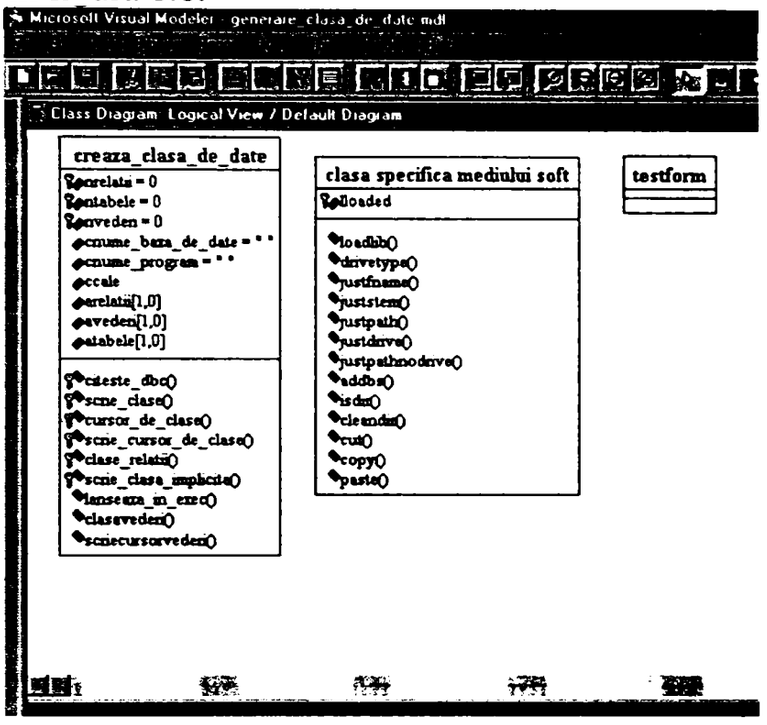


Fig 8.8 Clasele pentru crearea claselor de memorare a datelor realizate în Visual Modeler

Pornind de la baza de date relațională "Burghie" (figura 8.9), cu ajutorul claselor pentru generarea obiectelor de date, se ajunge la generarea codului obiectual, pentru clasa de date dorită.

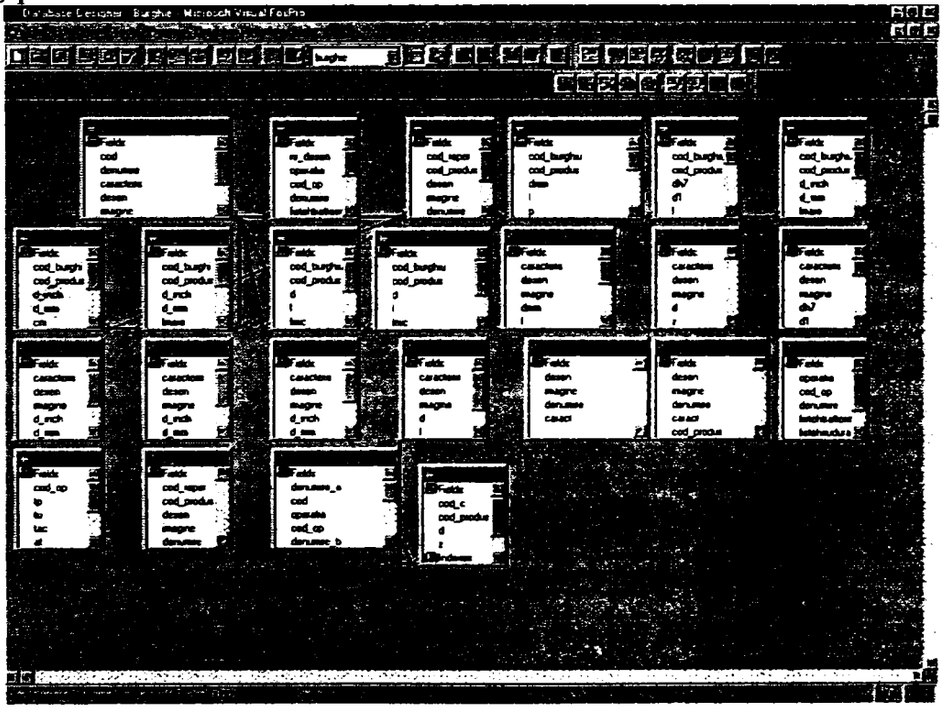
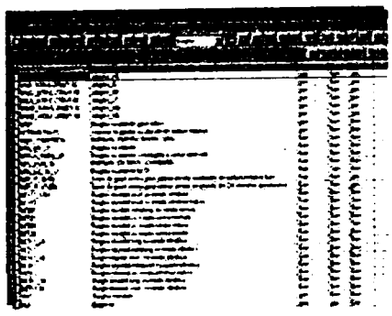
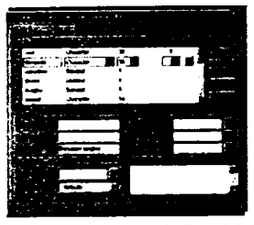


Fig 8.9 Baza de date relațională "Burghie"

Această bază de date relațională este formată din mai multe tabele şi vederi (figurile 8.10,8.11,8.12).

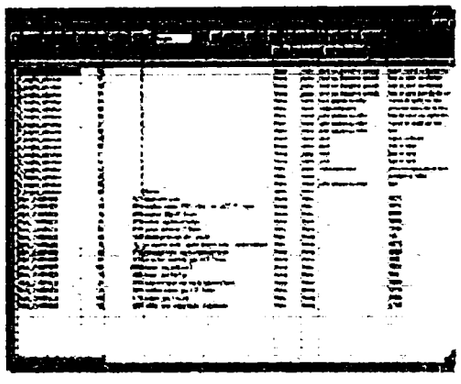


Continutul tabelei "Articole Burghiu"

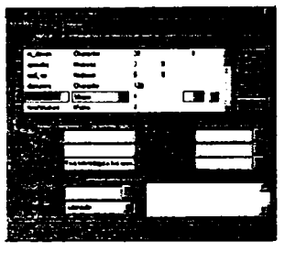


Structura tabelei "Articole Burghiu"

Fig 8.10 Tabela "Articole Burghiu"



Continutul tabelei "Operatii"



Structura tabelei "Operatii"

Fig 8.11 Tabela "Operații"

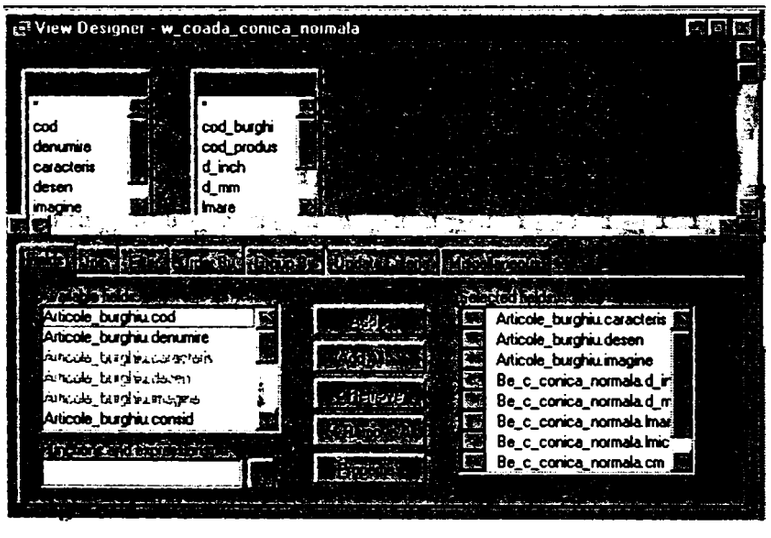


Fig 8.12 Vederea "W_coada_conica_normala"

Procesul de transformare este format din programul sursă de apelare a clasei de transformare, din clasa de transformare şi din programul sursă generat, care conține tabelele şi vederile transformate în clase de obiecte (figura 8.13).

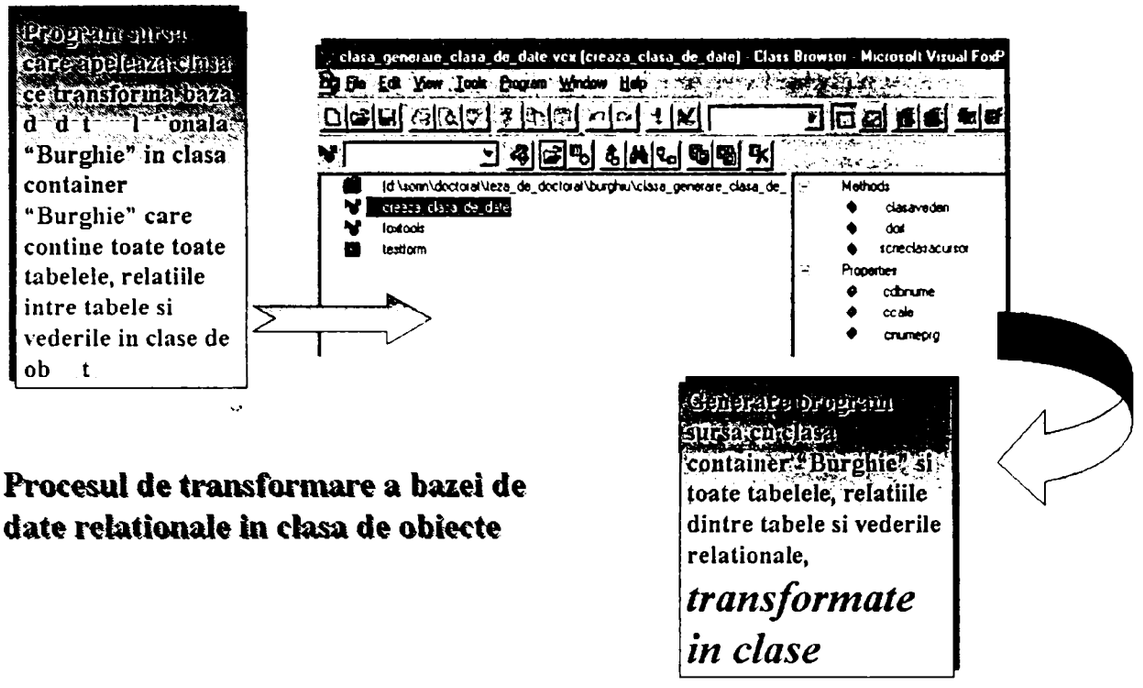


Fig 8.13 Procesul de transformare al bazei de date relaționale "Burghie" în clase de obiecte

Codul sursă al acestui proces este următorul:

Program pentru generarea bazei de date burghie

```
set classlib to clasa_generare_clasa_de_date
odbcgen=createobject("creaza_clasa_de_date")
odbcgen.cdbnume="burghie.dbc"
odbcgen.cnumeprg="baza_de_date_burghie.prg"
odbcgen.doit()
```

Clasa pentru generarea programului "Clasa_generare_clasa_de_date"

```
**************************************************
*-- Class:   creaza_clasa_de_date (d:\sorin\doctorat\teza_de_doctorat\burghiu\clasa_generare_clasa_de_date.vcx)
*-- ParentClass:  custom
*-- BaseClass:   custom
DEFINE CLASS creaza_clasa_de_date AS custom
        PROTECTED nrelatii
        nrelatii = 0
        PROTECTED ntabele
        ntabele = 0.
        PROTECTED nvederi
        nvederi = 0
        cdbnume = " "
        cnumeprg = " "
        Nume= "creaza_clasa_de_date"
        ccale = .F.
        PROTECTED arelatii[1]
        PROTECTED avederi[1]
        PROTECTED atabele[1]
        PROTECTED PROCEDURE citestebazadate
                        if !dbused(this.cdbnume)
                        ..      open database (this.cdbnume)
                        endif
                *local loftools
                *loftools=createobject("foxtools")
                *       this.ccale=loftools.justpath(dbc())
                *       this.cdbnume=loftools.justfname(dbc())
                        this.cdbnume="burghie.dbc"
                        this.ntabele=adbobjects(this.atabele,"table")
                        this.nvederi=adbobjects(this.avederi, "view")
```

```
                this.nrelatii=adbobjects(this.arelatii, "relation")
ENDPROC
PROTECTED PROCEDURE scrieclase
        set textmerge to (this.cnumeprg) noshow
        set textmerge on
        local lcoldcentury
        lcoldcentury=set ("century")
        set century on
        \*program. .....: <<this.cnumeprg>>
        \*baza de date.....: <<this.cdbnume>>
        \*generat in.....: <<mdy(date())+" - "+time()>>
        \#define databasepath "<<this.ccale>>"
        set textmerge off
        if this.ntabele>0
                this.clasecursor ()
        endif
        if this.nvederi >0
                this.clasavederi ()
        endif
        if this.nrelatii>0
                this.claserelatii()
        endif
        this.scrieclaseimplicite()
        set textmerge off
        set textmerge on
ENDPROC
PROTECTED PROCEDURE clasecursor
                local lncounter
                set textmerge on
                for lncounter=1 to this.ntabele
                        this.scrieclasacursor(this.atabele[lncounter])
                endfor
        *       for lncounter =1 to this.nvederi
        *               this.scrieclasacursor(this.atabele[lncounter])
        *       endfor
                set textmerge off
ENDPROC
PROTECTED PROCEDURE scrieclasacursor
        lparameters tcclassname
        \define class <<strtran(tcclassname,chr(32), "_")>> as cursor
        \  alias = "<<tcclassname>>"
        \  cursorsource="<<tcclassname>>"
        \  database = "<<this.cdbnume>>"
        \enddefine
        \

ENDPROC
PROTECTED PROCEDURE claserelatii
        lparameters tcclassname
        local  lncounter, lcclassname, lcchildalias, lcparentalias, lcchildorder, lcrelationalexpr
        set textmerge on
        for lncounter=1 to this.nrelatii
                lcclassnume= "relation" - alltrim(str(lncounter))
                lcchildalias = this.arelatii[lncounter,1]
                lcparentalias= this.arelatii[lncounter,2]
                lcchildorder= this.arelatii[lncounter,3]
                lcrelationalexpr= this.arelatii[lncounter,4]
                \define class <<lcclassname>> as relation
                \  childalias="<<lcchildalias>>"
                \  parentalias="<<lcparentalias>>"
                \  childorder="<<lcchildorder>>"
                \  relationalexpr="<<lcrelationalexpr>>"
                \enddefine
                \
        endfor
        set textmerge off
ENDPROC
PROTECTED PROCEDURE scrieclaseimplicite
        local laclasses[this.ntabele+this.nvederi+this.nrelatii]
        for lncounter=1 to this.ntabele
                laclasses[lncounter]=this.atabele[lncounter]
        endfor
        for lncounter=1 to this.nvederi
```

```
                laclasses[lncounter+this.ntabele]=this.avederi[lncounter]
            endfor
            for lncounter=1 to this.nrelatii
                laclasses[lncounter+this.ntabele+this.nvederi]="relations"+alltrim(str(lncounter))
            endfor
            set textmerge on
            \define class burghiu as dataenvironment
            for lncounter =1 to alen (laclasses,1)
                lcobjectname='o'+laclasses[lncounter]
                lcclassname=laclasses[lncounter]
            \add object <<lcobjectname>> as <<lcclassname>>
            endfor
            \enddefine
            \
            set textmerge off
        ENDPROC
        PROCEDURE doit
                if empty (this.cdbnume)
                    this.cdbnume=getfile("dbc", "va rugam selectati fisierul dbc pentru dump")
                    if !file(this.cdbnume)
                        = messagebox("nici un dbc selectat operatie abandonata",16)
                        return .f.
                    endif
                endif
                if empty (this.cnumeprg)
                    this.cnumeprg=putfile("PRG ce se creeaza"," ","PRG")
                    if empt(this.cnumeprg)
                        = messagebox("operatie abandonata",16)
                        return
                    endif
                endif
                if set("safety")="ON" and file (this.cnumeprg) and ;
                    messagebox("suprascrieti fisierul existent"+ alltrim(this.cnumeprg)+"?",36)#6
                    =messagebox("operatie abandonata")
                    return
                endif
                local lcoldsafety
                lcoldsafety = set ("safety")
                set safety off
                this.citestebazadate()
                this.scrieclase()
                set safety &lcoldsafety
        ENDPROC
        PROCEDURE clasavederi
            local lncounter
            set textmerge on
            for lncounter =1 to this.nvederi
                this.scriecursorvederi(this.avederi[lncounter])
            endfor
            set textmerge off
        ENDPROC
        PROCEDURE scriecursorvederi
            lparameters tcclassname
            \*acestea sunt clase vederi (VIEW)
            \define class <<strtran(tcclassname,chr(32), "_")>> as cursor
            \  alias = "<<tcclassname>>"
            \  cursorsource="<<tcclassname>>"
            \  database = "<<this.cdbnume>>"
            \enddefine
            \
        ENDPROC
ENDDEFINE
```

Baza de date "Burghie" generată prin program

```
*program.....: baza_de_date_burghie.prg
*baza de date.....: burghie.dbc
*generat in.....: March 05, 2002 - 19:27:03
#define databasepath ".F."
define class ARTICOLE_BURGHIU as cursor
  alias = "ARTICOLE_BURGHIU"
  cursorsource="ARTICOLE_BURGHIU"
  database = "burghie.dbc"
enddefine
define class OPERATII as cursor
  alias = "OPERATII"
  cursorsource="OPERATII"
  database = "burghie.dbc"
enddefine
define class REPERE_BURGHIU as cursor
  alias = "REPERE_BURGHIU"
  cursorsource="REPERE_BURGHIU"
  database = "burghie.dbc"
```

```
enddefine
define class COROANE_DE_GAURIRE as cursor
  alias = "COROANE_DE_GAURIRE"
  cursorsource="COROANE_DE_GAURIRE"
  database = "burghie dbc"
enddefine
define class SCULE_GAURIT_CU_PLACUTE as cursor
  alias = "SCULE_GAURIT_CU_PLACUTE"
  cursorsource="SCULE_GAURIT_CU_PLACUTE"
  database = "burghie.dbc"
enddefine
define class SC_GAURIT_FI_47_200 as cursor
  alias = "SC_GAURIT_FI_47_200"
  cursorsource="SC_GAURIT_FI_47_200"
  database = "burghie.dbc"
enddefine
define class B_E_SCURT_C_CIL as cursor
  alias = "B_E_SCURT_C_CIL"
  cursorsource="B_E_SCURT_C_CIL"
  database = "burghie dbc"
enddefine
define class BE_EXTRAL_C_CONICA as cursor
  alias = "BE_EXTRAL_C_CONICA"
  cursorsource="BE_EXTRAL_C_CONICA"
  database = "burghie.dbc"
enddefine
define class BE_C_CONICA_NORMALA as cursor
  alias = "BE_C_CONICA_NORMALA"
  cursorsource="BE_C_CONICA_NORMALA"
  database = "burghie.dbc"
enddefine
define class CU_PLACUTE_CM as cursor
  alias = "CU_PLACUTE_CM"
  cursorsource="CU_PLACUTE_CM"
  database = "burghie.dbc"
enddefine
define class GAURI_ADINCI_I as cursor
  alias = "GAURI_ADINCI_I"
  cursorsource="GAURI_ADINCI_I"
  database = "burghie.dbc"
enddefine
define class TIMP as cursor
  alias = "TIMP"
  cursorsource="TIMP"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_SC_GAURIT_CU_PLACUTE as cursor
  alias = "W_SC_GAURIT_CU_PLACUTE"
  cursorsource="W_SC_GAURIT_CU_PLACUTE"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_COROANE_GAURIRE as cursor
  alias = "W_COROANE_GAURIRE"
  cursorsource="W_COROANE_GAURIRE"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_SC_GAURIT_FI as cursor
  alias = "W_SC_GAURIT_FI"
  cursorsource="W_SC_GAURIT_FI"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_SCURT_C_CIL as cursor
  alias = "W_SCURT_C_CIL"
  cursorsource="W_SCURT_C_CIL"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_EXTRAL_COADA_CONICA as cursor
  alias = "W_EXTRAL_COADA_CONICA"
  cursorsource="W_EXTRAL_COADA_CONICA"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_COADA_CONICA_NORMALA as cursor
  alias = "W_COADA_CONICA_NORMALA"
  cursorsource="W_COADA_CONICA_NORMALA"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_GAURI_AD_I as cursor
  alias = "W_GAURI_AD_I"
  cursorsource="W_GAURI_AD_I"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_REPERE_G_ADINCI_I as cursor
  alias = "W_REPERE_G_ADINCI_I"
  cursorsource="W_REPERE_G_ADINCI_I"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_REPERE_CU_PLACUTE as cursor
  alias = "W_REPERE_CU_PLACUTE"
  cursorsource="W_REPERE_CU_PLACUTE"
  database = "burghie.dbc"
enddefine
*acestea sunt clase vederi (VIEW)
define class W_OPERATII as cursor
  alias = "W_OPERATII"
  cursorsource="W_OPERATII"
  database = "burghie.dbc"
enddefine
define class w_repere_dinti as cursor
  alias = "w_repere_dinti"
  cursorsource="w_repere_dinti"
  database = "burghie.dbc"
enddefine
define class w_fisa_tehnologica as cursor
  alias = "w_fisa_tehnologica"
  cursorsource="w_fisa_tehnologica"
  database = "burghie.dbc"
enddefine
define class burghiu as dataenvironment
add object oARTICOLE_BURGHIU as
ARTICOLE_BURGHIU
add object oOPERATII as OPERATII
add object oREPERE_BURGHIU as REPERE_BURGHIU
add object oCOROANE_DE_GAURIRE as
COROANE_DE_GAURIRE
add object oSCULE_GAURIT_CU_PLACUTE as
SCULE_GAURIT_CU_PLACUTE
add object oSC_GAURIT_FI_47_200 as
SC_GAURIT_FI_47_200
add object oB_E_SCURT_C_CIL as B_E_SCURT_C_CIL
add object oBE_EXTRAL_C_CONICA as
BE_EXTRAL_C_CONICA
add object oBE_C_CONICA_NORMALA as
BE_C_CONICA_NORMALA
add object oCU_PLACUTE_CM as CU_PLACUTE_CM
add object oGAURI_ADINCI_I as GAURI_ADINCI_I
add object oTIMP as TIMP
add object oW_SC_GAURIT_CU_PLACUTE as
W_SC_GAURIT_CU_PLACUTE
add object oW_COROANE_GAURIRE as
W_COROANE_GAURIRE
add object oW_SC_GAURIT_FI as W_SC_GAURIT_FI
add object oW_SCURT_C_CIL as W_SCURT_C_CIL
add object oW_EXTRAL_COADA_CONICA as
W_EXTRAL_COADA_CONICA
add object oW_COADA_CONICA_NORMALA as
W_COADA_CONICA_NORMALA
add object oW_GAURI_AD_I as W_GAURI_AD_I
```

```
add object oW_REPERE_G_ADINCI_I as W_REPERE_G_ADINCI_I
add object oW_REPERE_CU_PLACUTE as W_REPERE_CU_PLACUTE
add object oW_OPERATII as W_OPERATII
add object ow_repere_dinti as w_repere_dinti
add object ow_fisa_tehnologica as w_fisa_tehnologica
procedure vedeb
public varburghiu
        select(this.initialselectedalias)
        brow noappe nodelete nocdit
endproc
procedure alege
        select (this.initialselectedalias)
        requery()
endproc
enddefine
```

Acest mod de definire al claselor de date este valabil pentru toate părţile componente ale bazei de date relaţional obiectuale(figura 8.14).

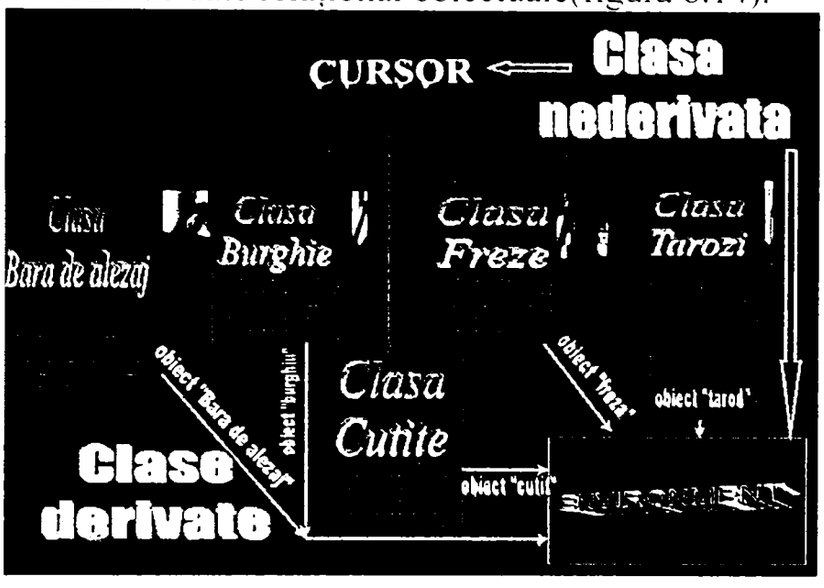


Fig 8.14 Clasele Sistemului de Fabricaţie al Sculelor

8.3 Definirea clasei generalizate pentru manipularea obiectelor de date din Sistemul de Fabricaţie al Sculelor

Aplicaţia care deserveşte SFD-ul este formată din mai multe părţi, funcţie de clasificarea acestuia. Pentru fiecare tip de sculă (freză, cuţit, etc) există clase pentru manipularea şi prezentarea informaţiilor. La baza tuturor acestor segmente ale aplicaţiei se găseşte un set de clase (figura 8.15), din care se obţin ulterior o serie de clase derivate necesare pentru realizarea aplicaţiei.

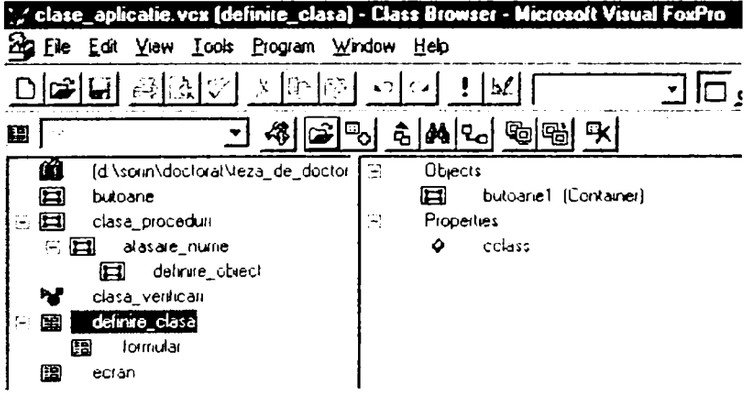


Fig 8.15 Clase de bază

Schema obiectuală a claselor de bază este prezentată în figura 8.16.

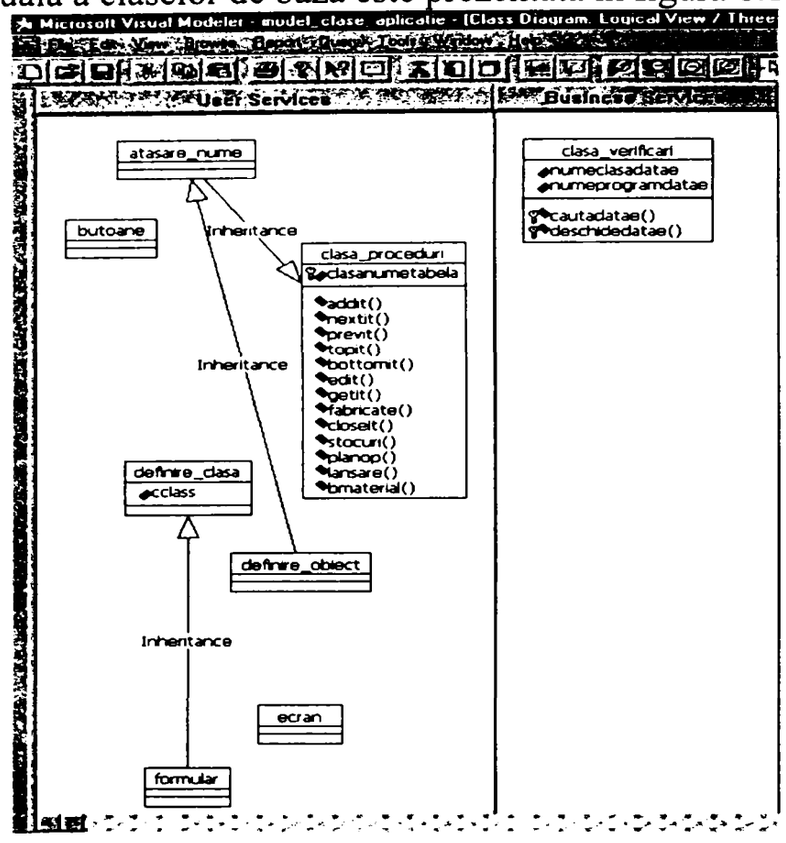


Fig 8.16 Schema obiectuală pentru clasele de bază

"Clasa_proceduri" de tip container conține toate procedurile necesare pentru manipularea şi vizualizarea informațiilor din clasa bază de date corespunzătoare unui anumit tip de sculă din cadrul SFD-ului. Totodată clasa conține o proprietate, care definește numele clasei pentru o anumită tabelă.

"Ataşare_nume" şi "Definire_obiect" sunt subclase ale "clasa_proceduri". Clasa "Ataşare_nume" conține un obiect ce aparține clasei "clasa_verificări" care permite definirea numelui clasei dataenvironment care conține obiectele cursor corespunzătoare tabelelor din baza de date relațională (numeclasadate). Proprietatea "numeprogramdate" permite definirea numelui programului care conține codul pentru definirea clasei de tip dataenvironment şi a claselor de tip cursor corespunzătoare tabelelor. Clasa "Definire_obiect" de tip container conține obiecte specifice pentru vizulizarea datelor din cadrul claselor de date (figura 8.17).

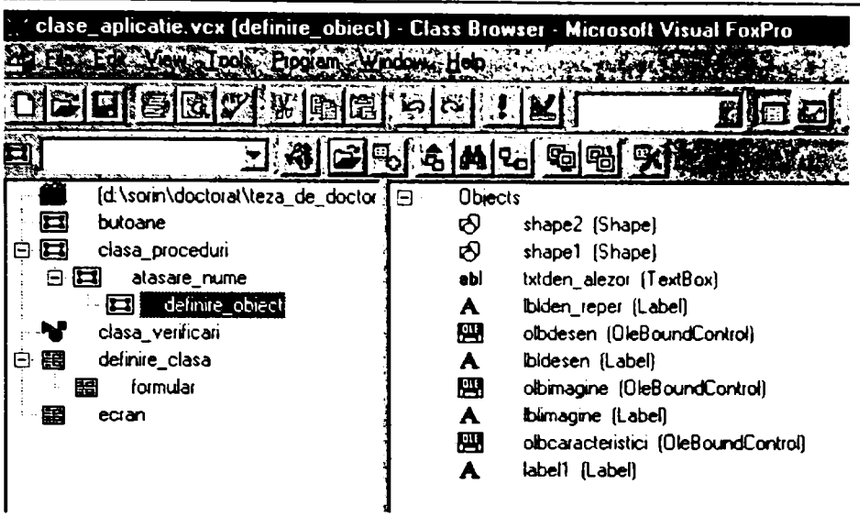


Fig 8.17 Clasa "Definire obiect"

"Clasa_verificări" conţine proprietăţile prin care este definit numele clasei dataenvironment care conţine toate obiectele de tip cursor ale unei baze de date, precum şi numele programului care conţine codul sursă al definiri claselor de date. "Definire_clasă" este o clasă care conţine un obiect butoane şi o proprietate care defineşte numele clasei din care vor fi manipulate datele. Aceasta este o clasă de tip cursor şi conţine date de o anumită natură. O subclasă a sa, este clasa "formular" de tip form care conţine un obiect "definire_obiect" pentru definirea ecranului aplicaţiei. Clasa "ecran" conţine un obiect specific clasei "butoane" fiind de tip form. Butoanele aplicaţiei sunt obiecte ale clasei "butoane".

Codul sursă al claselor prezentate este următorul:

```
*****************************************************
*-- Class:        def_clasa_si_program (d:\sorin\doctorat\teza_de_doctorat\burghiu\clasa_burghiu.vcx)
*-- ParentClass:  custom
*-- BaseClass:    custom
DEFINE CLASS def_clasa_si_program AS custom
        Height = 36
        Width = 36
        Name = "def_clasa_si_program"
        *-- numele clase data environment care se incarca
        cdenumeclasa = .F.
        *-- numele programului care contine clasa DE
        cdenumeprg = .F.
        *-- asigura ca s-a specificat un nume de clasa de mediu de date
        PROTECTED PROCEDURE chkcauta
                if type ("this.cdenumeclasa")#'C' or empty (this.cdenumeclasa)
                        =messagebox("Nu s-a specificat o clasa Data Environment. "+"Nu se poate instantia obiectul.",16, "eroare
de instantiere")
                        return .f.
                endif
        ENDPROC
        *-- deschide mediul de date
        PROTECTED PROCEDURE deschide
                public lodeb, lcclassname
                if !empty(this.cdenumeprg)
                        set procedure to (this.cdenumeprg) additive
                endif
                lcclassname=this.cdenumeclasa
                lodeb = createobject(lcclassname)
                if type("lodeb")# "O"
```

```
                        if !empty(this.cdenumeprg)
                                release procedure (this.cdenumeprg)
                        endif
                        return .f.
                endif
                lodeb.opentables()
                if !empty(this.cdenumeprg)
                        release procedure (this.cdenumeprg)
                endif
        ENDPROC
        PROCEDURE Init
                if !this.chkcauta()
                        return .f.
                endif

                return this.deschide()
        ENDPROC
ENDDEFINE
*
*-- EndDefine: def_clasa_si_program
```

Definirea clasei care conține procedurile pentru manipularea obiectelor de date.

```
DEFINE CLASS def_proceduri_burghiu AS container
        Width = 200
        Height = 200
        PROTECTED cnumetabela
        cnumetabela = "articole_burghiu"
        Name = "def_proceduri_burghiu"
        ADD OBJECT def_clasa_si_program1 AS def_clasa_si_program WITH ;
                Top = 12, ;
                Left = 12, ;
                Height = 48, ;
                Width = 36, ;
                cdenumeprg = "", ;
                Name = "Def_clasa_si_program1"
        PROCEDURE inceput
                select (this.cnumetabela)
                go top
                thisform.refresh()
                if type ("thisform")="o"
                        thisform.refresh()
                endif
        ENDPROC
        PROCEDURE sfarsit
                select(this.cnumetabela)
                go bottom
                        thisform.refresh()
                if type ("thisform")="o"
                        thisform.refresh()
                endif
        ENDPROC
        PROCEDURE nextit
                select(this.cnumetabela)
                skip 1
                if eof()
                        ??chr(7)
                        wait window nowait "la sfarsit de fisier"
                        go bottom
                endif
                        thisform.refresh()
        ENDPROC
        PROCEDURE previt
                select (this.cnumetabela)
                skip -1
                if bof()
                        ??chr(7)
                        wait window nowait "la inceput de fisier"
                        go top
                endif

                        thisform.refresh()
```

```
ENDPROC
PROCEDURE closeit
	oformburghiu.destroy
	release classlib flatbtn
	release classlib clasa_burghiu
	release all
	clear events
	close all
	thisform.visible=.f.
ENDPROC
PROCEDURE fabricateit
	public varburg,varprodusb
	do case
	case alltrim(oformburghiu.imagine_formular1.text1.value)="BURGHIU PENTRU GAURI LUNGI"
		lodeb.initialselectedalias="gauri_adinci_i"
		lodeb.vedeb()
		varburg=gauri_adinci_i.cod_burghiu
		varprodusb=gauri_adinci_i.cod_produs
	case alltrim(oformburghiu.imagine_formular1.text1.value)="Scule de gaurit pentru gauri adanci armate cu placute din
carburi metalice lipite"
		lodeb.initialselectedalias="scule_gaurit_cu_placute"
		lodeb.vedeb()
		varburg=scule_gaurit_cu_placute.cod_burghiu
		varprodusb=scule_gaurit_cu_placute.cod_produs
	case alltrim(oformburghiu.imagine_formular1.text1.value)="Scule de gaurit pentru gauri adanci armate cu placute din
CM elemente constructive"
		lodeb.initialselectedalias="sc_gaurit_fi_47_200"
		lodeb.vedeb()
		varburg=sc_gaurit_fi_47_200.cod_burghiu
		varprodusb=sc_gaurit_fi_47_200.cod_produs
	case alltrim(oformburghiu.imagine_formular1.text1.value)="Burghiu cu placute gauri adinci"
		lodeb.initialselectedalias="cu_placute_cm"
		lodeb.vedeb()
		varburg=cu_placute_cm.cod_burghiu
		varprodusb=cu_placute_cm.cod_produs
	endcase
ENDPROC
PROCEDURE repereit
	public vreperburghiu, vcodprodburg
		oformburghiu.imagine_formular1.btnrevin.visible=.t.
	do case
	case alltrim(oformburghiu.imagine_formular1.text1.value)="BURGHIU PENTRU GAURI LUNGI"
		lodeb.initialselectedalias="w_repere_g_adinci_i"
		lodeb.alege()
		oformburghiu.imagine_formular1.text1.controlsource="w_repere_g_adinci_i.denumire"
		oformburghiu.imagine_formular1.oleboundcontrol1.controlsource="w_repere_g_adinci_i.desen"
		oformburghiu.imagine_formular1.oleboundcontrol2.controlsource="w_repere_g_adinci_i.imagine"
		oformburghiu.imagine_formular1.cnumetabela="w_repere_g_adinci_i"
		thisform.refresh()
	case alltrim(oformburghiu.imagine_formular1.text1.value)="BURGHIU CU DINTI SCHIMBABILI"
		lodeb.initialselectedalias="w_repere_dinti"
		lodeb.alege()
		oformburghiu.imagine_formular1.text1.controlsource="w_repere_dinti.denumire"
		oformburghiu.imagine_formular1.oleboundcontrol1.controlsource="w_repere_dinti.desen"
		oformburghiu.imagine_formular1.oleboundcontrol2.controlsource="w_repere_dinti.imagine"
		oformburghiu.imagine_formular1.cnumetabela="w_repere_dinti"
		thisform.refresh()
	endcase
ENDPROC
PROCEDURE revinit
		oformburghiu.imagine_formular1.text1.controlsource="articole_burghiu.denumire"
		oformburghiu.imagine_formular1.oleboundcontrol1.controlsource="articole_burghiu.desen"
		oformburghiu.imagine_formular1.oleboundcontrol2.controlsource="articole_burghiu.imagine"
		oformburghiu.imagine_formular1.cnumetabela="articole_burghiu"
		oformburghiu.imagine_formular1.btnrevin.visible=.f.
		thisform.refresh()
ENDPROC
PROCEDURE fisa_tehnologicait
	do case
	case alltrim(oformburghiu.imagine_formular1.text1.value)="BURGHIU PENTRU GAURI LUNGI"
		lodeb.initialselectedalias="w_fisa_tehnologica"
		lodeb.alege()
```

```
                    oformburghiu2.imag_formular_21.text1.controlsource="w_fisa_tehnologica.denumire_a"
                *   oformburghiu.imagine_formular1.oleboundcontrol1.controlsource="w_repere_g_adinci_i.desen"
                *   oformburghiu.imagine_formular1.oleboundcontrol2.controlsource="w_repere_g_adinci_i.imagine"
                *   oformburghiu2.imag_formular_21.cnumetabela="w_fisa_tehnologica"
                *   requery()
                    oformburghiu2.visible=.t.
                    oformburghiu2.show()
                endcase
        ENDPROC
        PROCEDURE Init
                select(this.cnumetabela)
                if type ("thisform.cclass")#"U"
                        thisform.cclass=this.name
                endif
        ENDPROC
ENDDEFINE
```

Definirea clasei care determina numele clasei şi al programului în care se găsesc obiectele de date

```
DEFINE CLASS definire_numele_clasei_si_al_programului AS def_proceduri_burghiu
        Name = "definire_numele_clasei_si_al_programului"
        Def_clasa_si_program1.cdenumeprg = "baza_de_date_burghie"
        Def_clasa_si_program1.cdenumeclasa = "burghiu"
        Def_clasa_si_program1.Name = "Def_clasa_si_program1"
ENDDEFINE
```

Definirea clasei în care vor fi vizualizate datele

```
DEFINE CLASS imag_formular_2 AS definire_numele_clasei_si_al_programului
        Width = 1001
        Height = 746
        BackColor = RGB(221,221,221)
        cnumetabela = "w_fisa_tehnologica"
        Name = "imag_formular_2"
        Def_clasa_si_program1.Name = "Def_clasa_si_program1"
        ADD OBJECT label1 AS label WITH ;
                FontSize = 14, ;
                BackStyle = 0, ;
                Caption = "PRODUS", ;
                Height = 25, ;
                Left = 20, ;
                Top = 32, ;
                Width = 96, ;
                Name = "Label1"
        ADD OBJECT label2 AS label WITH ;
                FontSize = 14, ;
                BackStyle = 0, ;
                Caption = "COD", ;
                Height = 25, ;
                Left = 20, ;
                Top = 56, ;
                Width = 96, ;
                Name = "Label2"
        ADD OBJECT label3 AS label WITH ;
                FontBold = .T., ;
                FontItalic = .T., ;
                FontName = "Arial Black", ;
                FontShadow = .T., ;
                FontSize = 16, ;
                FontUnderline = .T., ;
                BackStyle = 0, ;
                Caption = "FISA TEHNOLOGICA DE PRELUCRARI", ;
                Height = 36, ;
                Left = 32, ;
                Top = 164, ;
                Width = 468, ;
                ForeColor = RGB(0,0,0), ;
                BackColor = RGB(255,255,255), ;
                Name = "Label3"
        ADD OBJECT label4 AS label WITH ;
                FontSize = 14, ;
                BackStyle = 0, ;
                Caption = "Buc/produs 1", ;
                Height = 25, ;
                Left = 536, ;
                Top = 176, ;
                Width = 156, ;
                Name = "Label4"
        ADD OBJECT shape1 AS shape WITH ;
                Top = 20, ;
                Left = 620, ;
                Height = 145, ;
                Width = 385, ;
                BackStyle = 0, ;
                BorderStyle = 1, ;
                BorderWidth = 2, ;
                Name = "Shape1"
        ADD OBJECT line1 AS line WITH ;
                BorderWidth = 2, ;
                Height = 1, ;
                Left = 620, ;
                Top = 68, ;
                Width = 384, ;
                Name = "Line1"
        ADD OBJECT line2 AS line WITH ;
                BorderWidth = 2, ;
                Height = 1, ;
                Left = 620, ;
                Top = 116, ;
                Width = 384, ;
                Name = "Line2"
        ADD OBJECT line3 AS line WITH ;
                BorderWidth = 2, ;
                Height = 144, ;
                Left = 728, ;
                Top = 20, ;
                Width = 0, ;
                Name = "Line3"
        ADD OBJECT line4 AS line WITH ;
                BorderWidth = 2, ;
                Height = 144, ;
                Left = 812, ;
                Top = 20, ;
                Width = 0, ;
                Name = "Line4"
        ADD OBJECT line5 AS line WITH ;
                BorderWidth = 2, ;
```

```
            Height = 144, ;
            Left = 908, ;
            Top = 20, ;
            Width = 0, ;
            Name = "Line5"
ADD OBJECT label5 AS label WITH ;
            FontName = "Arial Black", ;
            FontSize = 14, ;
            BackStyle = 0, ;
            Caption = "DATA", ;
            Height = 36, ;
            Left = 740, ;
            Top = 32, ;
            Width = 60, ;
            Name = "Label5"
ADD OBJECT label6 AS label WITH ;
            FontName = "Arial Black", ;
            FontSize = 14, ;
            BackStyle = 0, ;
            Caption = "SEMNAT", ;
            Height = 36, ;
            Left = 812, ;
            Top = 32, ;
            Width = 96, ;
            Name = "Label6"
ADD OBJECT label7 AS label WITH ;
            FontName = "Arial Black", ;
            FontSize = 14, ;
            BackStyle = 0, ;
            Caption = "SIMB", ;
            Height = 36, ;
            Left = 920, ;
            Top = 32, ;
            Width = 60, ;
            Name = "Label7"
ADD OBJECT label8 AS label WITH ;
            FontName = "Arial Black", ;
            FontSize = 12, ;
            BackStyle = 0, ;
            Caption = "VERIFICAT", ;
            Height = 36, ;
            Left = 632, ;
            Top = 80, ;
            Width = 96, ;
            Name = "Label8"
ADD OBJECT label9 AS label WITH ;
            FontName = "Arial Black", ;
            FontSize = 12, ;
            BackStyle = 0, ;
            Caption = "MODIFICAT", ;
            Height = 36, ;
            Left = 632, ;
            Top = 128, ;
            Width = 96, ;
            Name = "Label9"
ADD OBJECT label10 AS label WITH ;
            FontSize = 14, ;
            BackStyle = 0, ;
            Caption = "LA", ;
            Height = 25, ;
            Left = 20, ;
            Top = 8, ;
            Width = 36, ;
            Name = "Label10"
ADD OBJECT text4 AS textbox WITH ;
            Height = 25, ;
            Left = 740, ;
            Top = 80, ;
            Width = 61, ;
            Name = "Text4"
ADD OBJECT text5 AS textbox WITH ;
            Height = 25, ;
            Left = 824, ;
            Top = 80, ;
            Width = 73, ;
            Name = "Text5"
ADD OBJECT text6 AS textbox WITH ;
            Height = 25, ;
            Left = 920, ;
            Top = 80, ;
            Width = 73, ;
            Name = "Text6"
ADD OBJECT text7 AS textbox WITH ;
            Height = 25, ;
            Left = 740, ;
            Top = 128, ;
            Width = 61, ;
            Name = "Text7"
ADD OBJECT text8 AS textbox WITH ;
            Height = 25, ;
            Left = 824, ;
            Top = 128, ;
            Width = 73, ;
            Name = "Text8"
ADD OBJECT text9 AS textbox WITH ;
            Height = 25, ;
            Left = 920, ;
            Top = 128, ;
            Width = 73, ;
            Name = "Text9"
ADD OBJECT command1 AS commandbutton
WITH ;
            Top = 596, ;
            Left = 140, ;
            Height = 49, ;
            Width = 109, ;
            FontSize = 8, ;
            WordWrap = .T., ;
            Caption = "FISA TEHNOLOGICA DE
TRATAMENT TERMIC", ;
            Name = "Command1"

ADD OBJECT command2 AS commandbutton
WITH ;
            Top = 596, ;
            Left = 776, ;
            Height = 49, ;
            Width = 109, ;
            FontSize = 8, ;
            WordWrap = .T., ;
            Caption = "FISA TEHNOLOGICA DE
SUDURA", ;
            Name = "Command2"
ADD OBJECT btnback AS commandbutton WITH ;
            Top = 596, ;
            Left = 468, ;
            Height = 49, ;
            Width = 97, ;
            Caption = "BACK", ;
            Name = "btnback"
ADD OBJECT text3 AS textbox WITH ;
            FontBold = .T., ;
            FontSize = 12, ;
            Alignment = 3, ;
            Value = (date()), ;
            Height = 25, ;
            Left = 68, ;
            ReadOnly = .T., ;
            Top = 8, ;
            Width = 120, ;
            DisabledBackColor = RGB(255,255,255).
;
            Name = "Text3"
```

```
        ADD OBJECT text1 AS textbox WITH ;
                FontBold = .T., ;
                ControlSource = "", ;
                Height = 25, ;
                Left = 128, ;
                ReadOnly = .T., ;
                Top = 32, ;
                Width = 324, ;
                DisabledBackColor = RGB(255,255,255),

                BorderColor = RGB(0,0,0), ;
                Name = "Text1"
        ADD OBJECT text2 AS textbox WITH ;
                FontBold = .T., ;
                ControlSource =
"w_fisa_tehnologica.cod", ;
                Height = 25, ;
                Left = 128, ;
                ReadOnly = .T., ;
                Top = 56, ;
                Width = 324, ;
                DisabledBackColor = RGB(255,255,255),

                DisabledForeColor = RGB(192,192,192),

                Name = "Text2"
        ADD OBJECT grid1 AS grid WITH ;
                ColumnCount = 10, ;
                FontBold = .T., ;
                DeleteMark = .F., ;
                GridLines = 3, ;
                GridLineWidth = 1, ;
                HeaderHeight = 20, ;
                Height = 348, ;
                Left = 12, ;
                Panel = 1, ;
                ReadOnly = .T., ;
                RecordMark = .F., ;
                RecordSource = "w_fisa_tehnologica", ;
                ScrollBars = 3, ;
                Top = 228, ;
                Width = 1020, ;
                ForeColor = RGB(0,0,0), ;
                BackColor = RGB(255,255,255), ;
                GridLineColor = RGB(0,0,0), ;
                Name = "Grid1", ;
        ADD OBJECT fttermic AS editbox WITH ;
                Height = 372, ;
                Left = 12, ;
                Top = 228, ;
                Visible = .F., ;
                Width = 624, ;
                ControlSource =
"w_fisa_tehnologica.fistehtratterm", ;
                Name = "FTTERMIC"
        ADD OBJECT ftsudura AS editbox WITH ;
                Height = 372, ;
                Left = 396, ;
                Top = 228, ;
                Visible = .F., ;
                Width = 624, ;
                ControlSource =
"w_fisa_tehnologica.fistehsudura", ;
                Name = "ftsudura"
        PROCEDURE command1.Click

        oformburghiu2.imag_formular_21.fttermic.visible=.t.
        ENDPROC
        PROCEDURE command1.RightClick

        oformburghiu2.imag_formular_21.fttermic.visible=.f.
        ENDPROC
        PROCEDURE command2.Click

        oformburghiu2.imag_formular_21.ftsudura.visible=.t.
        ENDPROC
        PROCEDURE command2.RightClick

        oformburghiu2.imag_formular_21.ftsudura.visible=.f

        ENDPROC
        PROCEDURE btnback.Click
                thisform.visible=.f.
        ENDPROC
ENDDEFINE
```

Definire clasa imagine formular

```
DEFINE CLASS imagine_formular AS
definire_numele_clasei_si_al_programului
        Width = 1017
        Height = 746
        Visible = .T.
        Name = "imagine_formular"
        Def_clasa_si_program1.Name =
"Def_clasa_si_program1"
        ADD OBJECT text1 AS textbox WITH ;
                ControlSource =
"articole_burghiu.denumire", ;
                Height = 24, ;
                Left = 12, ;
                Top = 36, ;
                Width = 420, ;
                Name = "Text1"
        ADD OBJECT oleboundcontrol1 AS
oleboundcontrol WITH ;
                Top = 12, ;
                Left = 456, ;
                Height = 420, ;
                Width = 493, ;
                ControlSource =
"articole_burghiu.desen", ;
                Stretch = 1, ;
                Name = "Oleboundcontrol1"
        ADD OBJECT oleboundcontrol2 AS
oleboundcontrol WITH ;
                Top = 84, ;
                Left = 12, ;
                Height = 252, ;
                Width = 336, ;
                ControlSource =
"articole_burghiu.imagine", ;
                Stretch = 1, ;
                Name = "Oleboundcontrol2"
        ADD OBJECT btnrevin AS commandbutton WITH ;
                Top = 480, ;
                Left = 888, ;
                Height = 49, ;
                Width = 85, ;
                Caption = "Back", ;
                Visible = .F., ;
                Name = "btnrevin"
        PROCEDURE btnrevin.Click
                local lcclassname
                lcclassname=thisform.cclass
                thisform.&lcclassname..revinit()
        ENDPROC
ENDDEFINE
```

Definirea clasei pentru procedurile butoanelor

```
DEFINE CLASS definire_clasa_pt_proceduri_butoane AS form
        DataSession = 2
        Top = 0
        Left = 0
```

```
        Height = 250
        Width = 946
        DoCreate = .T.
        Caption = "form"
        BackColor = RGB(128,128,128)
        Name = "definire_clasa_pt_proceduri_butoane"
        *-- defineste clasa catre care se face navigarea
        cclass = .F.
        ADD OBJECT butoane1 AS butoane WITH ;
                Top = 60, ;
                Left = 12, ;
                Width = 744, ;
                Height = 84, ;
                Name = "Butoane1", ;
                Timer_flat1.Name = "Timer_flat1", ;
                butclose.Shadow.Name = "Shadow", ;
                butclose.Line4.DefLeft = "", ;
                butclose.Line4.DefHeight = "", ;
                butclose.Line4.Name = "Line4", ;
                butclose.Line2.DefWidth = "", ;
                butclose.Line2.Name = "Line2", ;
                butclose.Line1.DefTop = "", ;
                butclose.Line1.DefWidth = "", ;
                butclose.Line1.Name = "Line1", ;

        ADD OBJECT imag_formular_21 AS
imag_formular_2 WITH ;
                Top = 0, ;
                Left = 0, ;
                Width = 1008, ;
                Height = 744, ;
                Name = "Imag_formular_21", ;
                Def_clasa_si_program1.Name =
"Def_clasa_si_program1", ;
                Label1.Name = "Label1", ;
                Text1.Name = "Text1", ;
                Label2.Name = "Label2", ;
                Text2.Name = "Text2", ;
                Text3.Name = "Text3", ;
                Label3.Name = "Label3", ;
                Grid1.Column1.Header1.Name =
"Header1", ;
                Grid1.Column1.Text1.Name = "Text1", ;
                Grid1.Column1.Name = "Column1", ;
                Grid1.Column2.Header1.Name =
"Header1", ;
                Grid1.Column2.Text1.Name = "Text1", ;
                Grid1.Column2.Name = "Column2", ;
                Grid1.Column3.Header1.Name =
"Header1", ;
                Grid1.Column3.Text1.Name = "Text1", ;
                Grid1.Column3.Name = "Column3", ;
                Grid1.Column4.Header1.Name =
"Header1", ;
                Grid1.Column4.Text1.Name = "Text1", ;
                Grid1.Column4.Name = "Column4", ;
                Grid1.Column5.Header1.Name =
"Header1", ;
                Grid1.Column5.Text1.Name = "Text1", ;
                Grid1.Column5.Name = "Column5", ;
                Grid1.Column6.Header1.Name =
"Header1", ;
                Grid1.Column6.Text1.Name = "Text1", ;
                Grid1.Column6.Name = "Column6", ;
                Grid1.Column7.Header1.Name =
"Header1", ;
                Grid1.Column7.Text1.Name = "Text1", ;
                Grid1.Column7.Name = "Column7", ;
                Grid1.Column8.Header1.Name =
"Header1", ;
                Grid1.Column8.Text1.Name = "Text1", ;
                Grid1.Column8.Name = "Column8", ;
                Grid1.Column9.Header1.Name =
"Header1", ;
                Grid1.Column9.Text1.Name = "Text1", ;
                Grid1.Column9.Name = "Column9", ;
                Grid1.Column10.Header1.Name =
"Header1", ;
                Grid1.Column10.Text1.Name = "Text1", ;
                Grid1.Column10.Name = "Column10", ;
                Grid1.Name = "Grid1", ;
                Label4.Name = "Label4", ;
                Shape1.Name = "Shape1", ;
                Line1.Name = "Line1", ;
                Line2.Name = "Line2", ;
                Line3.Name = "Line3", ;
                Line4.Name = "Line4", ;
                Line5.Name = "Line5", ;
                Label5.Name = "Label5", ;
                Label6.Name = "Label6", ;
                Label7.Name = "Label7", ;
                Label8.Name = "Label8", ;
                Label9.Name = "Label9", ;
                Label10.Name = "Label10", ;
                Text4.Name = "Text4", ;
                Text5.Name = "Text5", ;
                Text6.Name = "Text6", ;
                Text7.Name = "Text7", ;
                Text8.Name = "Text8", ;
                Text9.Name = "Text9", ;
                Command1.Name = "Command1", ;
                Command2.Name = "Command2", ;
                FTTERMIC.Name = "FTTERMIC", ;
                ftsudura.Name = "ftsudura", ;
                btnback.Name = "btnback"
ENDDEFINE
```

Definirea clasei formular burghiu

```
DEFINE CLASS formular_burghiu AS
definire_clasa_pt_proceduri_butoane
        Top = 0
        Left = 0
        Height = 742
        Width = 1016
        DoCreate = .T.
        Caption = "BURGHIU"
        Name = "formular_burghiu"
        Butoane1.Timer_flat1.Name = "Timer_flat1"
        Butoane1.butclose.Shadow.Name = "Shadow"
        Butoane1.butclose.Line4.DefLeft = ""
        Butoane1.butclose.Line4.DefHeight = ""
        Butoane1.butclose.Line4.Name = "Line4"
        Butoane1.butclose.Line2.DefWidth = ""
        Butoane1.butclose.Line2.Name = "Line2"
        Butoane1.butclose.Line1.DefTop = ""
        Butoane1.butclose.Line1.DefWidth = ""
        Butoane1.butclose.Line1.Name = "Line1"
        Butoane1.butclose.Line3.DefHeight = ""
        Butoane1.butclose.Line3.Name = "Line3"
        Butoane1.Top = 468
        Butoane1.Left = 24
        Butoane1.Width = 744
        Butoane1.Height = 84
        Butoane1.ZOrderSet = 1
        Butoane1.Name = "Butoane1"
        ADD OBJECT imagine_formular1 AS
imagine_formular WITH ;
                Top = 0, ;
                Left = 12, ;
                Width = 1008, ;
```

```
                Height = 744, ;
                ZOrderSet = 0, ;
                Name = "Imagine_formular1", ;
                Def_clasa_si_program1.Name =
"Def_clasa_si_program1", ;
                Text1.Name = "Text1", ;
                Oleboundcontrol1.Top = 12, ;
                Oleboundcontrol1.Left = 456, ;
                Oleboundcontrol1.Height = 420, ;
                Oleboundcontrol1.Width = 493, ;
                Oleboundcontrol1.Name =
"Oleboundcontrol1", ;
                Oleboundcontrol2.Top = 84, ;
                Oleboundcontrol2 Left = 12, ;
                Oleboundcontrol2.Height = 252, ;
                Oleboundcontrol2.Width = 336, ;
                Oleboundcontrol2.Name =
"Oleboundcontrol2", ;
                btnrevin.Name = "btnrevin"
        ADD OBJECT timer_flat1 AS timer_flat WITH ;
                Top = 432, ;
                Left = 24, ;
                Height = 36, ;
                Width = 48, ;
                Name = "Timer_flat1"
ENDDEFINE
*
```

Realizarea claselor prezentate anterior permite abordarea pur obiectuală a SFS-ului atât din punct de vedere al memorării datelor cât şi din punctul de vedere al gestionării acestora. Se realizează următoarele activităţi:

- Baza de date relaţională este transformată într-o bază de date obiectuală. Transformarea se realizează obiectual cu ajutorul clasei "creaza_clasa_de_date". Codul program rezultat este folosit pentru gestionarea datelor fiind definită o clasă de tip dataenvironment, care conţine mai multe obiecte corespunzătoare tabelelor din baza de date relaţională, fiecare tabel fiind o clasă distinctă de tip cursor. Fiecare tuplă din cadrul tabelelor este considerată un obiect individual cu proprietăţile specifice structurii tabelului;
- Toate procedurile necesare gestionării datelor se găsesc într-o clasă specifică. Aplicarea unei proceduri înseamnă de fapt, apelarea metodei cu acelaşi nume a clasei;
- Structura arborescentă de clase prin care se definesc obiectele necesare vizualizării datelor, a celor necesare pentru consultarea datelor se face într-un mod nou. Abordarea obiectuală, permite încă din faza de proiectare a aplicaţiei pentru SFS, facilităţi importante privind înţelegerea legăturii dintre mediile de stocare şi gestiunea daelor;
- Un alt aspect important este faptul că, prin folosirea conceptului de clasă derivată, clasele de bază folosite pentru gestiunea datelor pot fi aplicate la fiecare modul al SFS-ului.

8.4 Definirea claselor derivate şi a schemelor obiectuale pentru Sistemul de Fabricaţie al Sculelor

8.4.1 Definirea claselor derivate

Proiectarea şi ulterior implementarea bazelor de date pentru SFS s-a făcut pe baza teoriei dezvoltate în capitolele 4 şi 5. Clasele de bază sunt considerate atât tipurile de clase proprii mediului de programare cât şi clase noi. Respectând definiţia care afirmă că o clasă derivată, este o clasă care care se poate obţine dintr-o clasă de bază pe baza unei interogări, pentru fiecare tip de bază de date s-au obţinut clase derivate. În general clasele derivate sau obţinut prin moştenire

ele păstrând proprietăţile claselor din care provin. Semantica acestor clase este obiect generat, deoarece pentru orice tip al clasei de bază, clasa generată va conţine obiecte noi. Funcţie de fiecare tip de bază de date, s-au analizat proprietăţile claselor specifice şi s-au elaborat schemele conceptuale şi schemele externe corespunzătoare. Pentru baza de date realizată sa păstrat arhitectura ANSI/SPARC specifică bazelor de date relaţionale, adaptată într-un mod nou pentru baza de date obiectuală.

Pentru clasa "Alezoare" care este de tip predefinit *dataenvironment* şi care conţine mai multe obiecte ale claselor, corespunzătoare tabelelor de tip cursor s-au definit clasele derivate, schema conceptuală şi schema externă.

Clasele de bază s-au considerat clasa cursor, alezoare de mână, alezoare de maşină. Pentru clasele care au proprietăţi în comun, s-a realizat închiderea prin moştenire (figura 8.18).

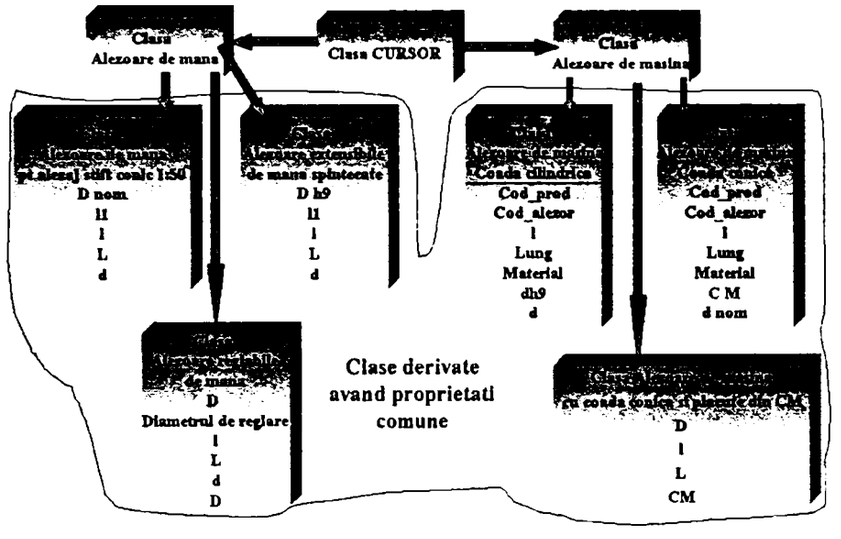


Fig 8.18 Clase de bază şi clase derivate pentru schema obiectuală "Alezoare de mână"

O altă modalitate de a obţine clase derivate, constă în obţinerea vederilor din clasele de bază. Acestea vor fi clase de tip cursor, şi au proprietăţi în comun cu clasele din care sunt derivate (figura 8.19).

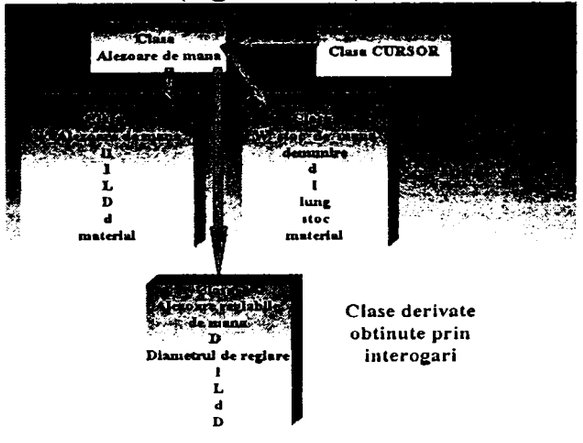


Fig 8.19 Clase derivate obţinute ca vederi derivate

Această abordare este valabilă pentru toate modulele SFS-ului. Mai mult decât atât, fiecare tabelă din baza de date este considerată un obiect care aparține clasei alezoare. Datorită acestui fapt nivelul de derivabilitate poate fi interpretat atât la nivel superior cât şi la nivel inferior, putând fi create noi proprietăți ale claselor, procesul de derivabilitate putând fi dezvoltat funcție de necesitățile sistemului de fabricație. Schema conceptuală şi schema externă pot fi dezvoltate permanent funcție de noile poduse care intră în procesul de fabricație sau funcție de unele produse care sunt scoase din procesul de fabricație.

Clasele derivate trebuie definite în dicționarul de date. Acest lucru se realizează automat în cadrul mediului soft cu care se lucrează. Derivabilitatea claselor poate fi extinsă şi la clasele care realizează gestionarea datelor în cadrul aplicației. Pornind de la clasele de bază pentru gestionarea datelor, pentru fiecare modul din clasificarea sculelor se vor realiza clase derivate care au proprietăți asemănătoare. Clasele pentru gestionarea datelor din diverse clase *cursor* sunt asemănătoare, numai proprietățile prin care se modifică numele clasei, numele procedurilor şi numele tabelei, se schimbă. Clasa butoane este aceeaşi pentru toate modulele SFS-ului. Aceste clase pot fi considerate clase derivate din clasele de bază (figura 8.20).

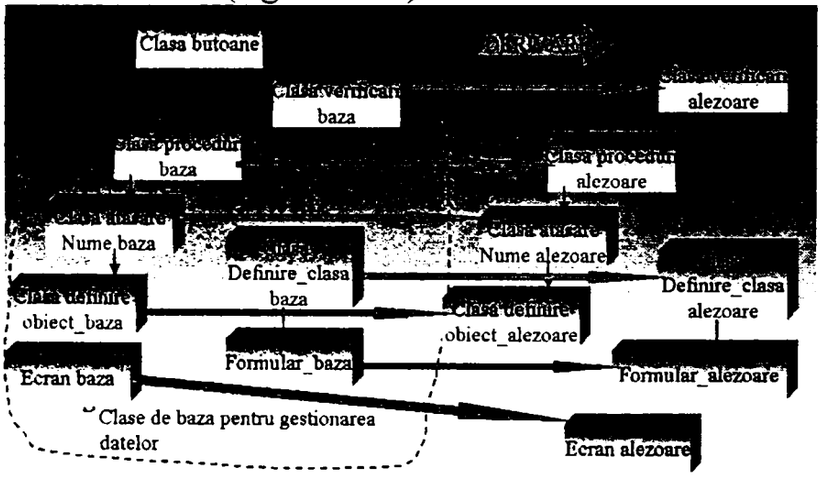


Fig 8.20 Procesul de derivare al claselor pentru schema obiectuală "Alezoare"

Procesul de derivabilitate poate continua în cadrul bazei de date "Alezoare". Astfel relațiile pot fi "sparte" în mai multe relații, fiecare relație având un anumit număr de atribute. Fiecare relație obținută este definită ca şi clasă de obiecte. Fiecare tuplă este considerată ca şi un obiect, identificabil în dicționarul de date. Identificarea obiectului se poate realiza prin atribut de tip cheie primară, fiecare obiect având în o valoare distinctă în atributul respectiv. Prin semantica obiect generat se pot obține clase noi. Această clasă reprezintă aceleaşi concepte ca şi clasa inițială dar adaptate la noile cerințe impuse de atributul sau atributele noi. Important este de reținut faptul că se poate definii un

concept nou, prin agregarea obiectelor obţinute prin transformarea valorilor în obiecte.

Iniţial nu există clasa "Alezoare", care conţine dimensiunile acesteia. Definim această clasă şi din ea se definesc clasele derivate "Dimensiuni" şi "Dimensiuni1" prin semantica obiect generat. Clasa "Alezoare1" este derivată prin semantica obiect conservat. Această clasă reprezintă aceleaşi concepte ca şi clasa iniţială "Alezoare", dar adaptate noilor cerinţe impuse de către clasa "Dimensiuni", cu care este legată printr-o relaţie de agregare (figura 8.21).

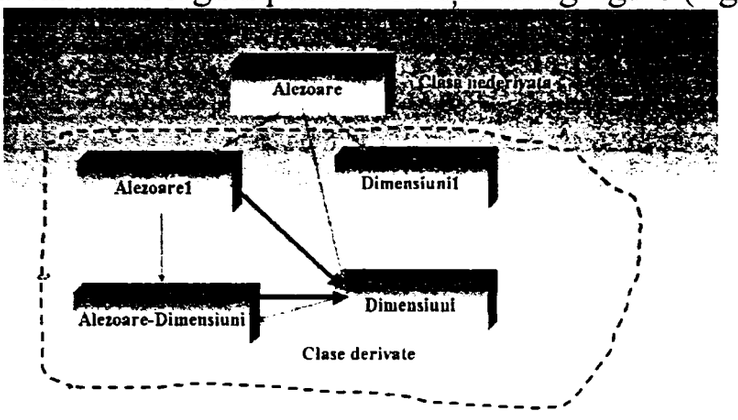


Fig 8.21 Clase derivate prin semantica obiect generat

Fiecare obiect de bază sau obiect derivat este reprezentat prin indicatorul său. Considerăm reprezentarea sub formă de relaţie a clasei "Alezoare", în care fiecare tuplă reprezintă un obiect identificat printr-un cod, numit cod de identificare obiect tabelul 1.

Tabelul 1

Cod de identificare obiect	Denumire	Dimensiuni
A1	De mană	Dm6, l, L, d
A2	De mană coadă cilindrică	Dm6, l, L, dh9
A3	De maşină coadă conică cu plăcuţe	Dm6, l, L, CM
A4	De mână pentru alezaj ştift conic	Dnom, L1, L, d
A5	Pentru găuri nit	Dnom, L, l, l1, CM
A6	Canal elicoidal coadă cilindrică	Dnom, l1,l, L, d

Fiecare identificator al clasei "Alezoare1" este identic cu identificatorul corespuzător din clasa nederivată (tabelul 2).

Tabelul 2

Cod de identificare obiect	Obiecte de bază
A1	A1

A2	A2
A3	A3
A4	A4
A5	A5
A6	A6

Clasa "Dimensiuni" corespunzătoare clasei "Alezoare" este generată de aşa manieră, încât obiectul este generat pentru fiecare valoare diferită a proprietăţilor clasei "Dimensiuni". Astfel valorile sunt transformate în obiecte (tabelul 3).

Tabelul 3

Cod de identificare obiect	Atribut	Obiect de bază	Denumire
D11	Dm6	{A1,A2,A3}	diammetric 6
D12	l	{A1,A2,A3,A5,A6}	lmic
D13	L	{A1,A2,A3,A4,A5,A6}	lmare
D14	d	{A1,A4,A6}	dmic
D15	dh9	{A2}	diam
D16	Dnom	{A4,A5,A6}	diamnominal
D17	CM	{A3,A5}	placute_carburi_metalice
D18	L1	{A4}	lmare1
D19	l1	{A5,A6}	lmic1

Dacă există două sau mai multe obiecte ale clasei "Alezoare", care au aceeaşi valoare returnată de proprietăţile clasei "Dimensiuni" atunci toate obiectele sunt obiecte de bază.

Clasa derivată "Dimensiuni1" este reprezentată în tabelul 4.

Tabelul 4

Cod de identificare obiect	Atribut	Obiect de bază	Denumire
E10	Dm6,A1	{A1,A2,A3}	diammetric6
E11	Dm6,A2	{A1,A2,A3}	diammetric6
E12	Dm6,A3	{A1,A2,A3}	diammetric6
E13	l,A1	{A1,A2,A3,A5,A6}	lmic
E14	l,A2	{A1,A2,A3,A5,A6}	lmic
E15	l,A3	{A1,A2,A3,A5,A6}	lmic
E16	l,A5	{A1,A2,A3,A5,A6}	lmic
E17	l,A6	{A1,A2,A3,A5,A6}	lmic
E18	L,A1	{A1,A2,A3,A4,A5,A6}	lmare
E19	L,A2	{A1,A2,A3,A4,A5,A6}	lmare
E20	L,A3	{A1,A2,A3,A4,A5,A6}	lmare

E21	L,A4	{A1,A2,A3,A4,A5,A6}	lmare
E22	L,A5	{A1,A2,A3,A4,A5,A6}	lmare
E23	L,A6	{A1,A2,A3,A4,A5,A6}	lmare
E24	dh9,A2	{A2}	diam
E25	dnom,A4	{A4,A5,A6}	dnom
E26	dnom,A5	{A4,A5,A6}	dnom
E27	dnom,A6	{A4,A5,A6}	dnom
E28	Cm,A3	{A3,A5}	plăcuțe carburi metalice
E29	Cm,A5	{A3,A5}	plăcuțe carburi metalice
E30	L1,A4	{A4}	lmare1
E31	l1,A5	{A5,A6}	lmic1
E32	l1,A6	{A5,A6}	lmic1

Clasa Alezoare1 a fost definită prin semantica obiect conservat, un singur atribut intern fiind folosit pentru identificarea obiectului de bază (tabelul 5).

Tabelul 5

Cod de identificare obiect	Atribut	Obiect de bază	Identificator "Dimensiuni"
A1	A1	A1	D11,D12,D13,D14
A2	A2	A2	D11,D12,D13,D15
A3	A3	A3	D11,D12,D13,D17
A4	A4	A4	D13,D14,D16,D18
A5	A5	A5	D12,D13,D16,D17,D19
A6	A6	A6	D12,D13,D14,D16,D19

Clasa "Alezoare-Dimensiuni" a fost definită prin semantica obiect generat folosind asocierea obiectelor (tabelul 6).

Tabelul 6

Cod de identificare obiect	Atribut	Obiecte de baza	Dimens	Alezor
AD11	D11,{A1,A2,A3}	D11,{A1,A2,A3}	D11	{A1,A2,A3}
AD12	D12,{A1,A2,A3,A5,A6}	D12,{A1,A2,A3,A5,A6}	D12	{A1,A2,A3, A5,A6}
AD13	D13, {A1,A2,A3,A5,A6}	D13, {A1,A2,A3,A5,A6}	D13	{A1,A2,A3, A5,A6}
AD14	D14,{A1,A4,A6}	D14,{A1,A4,A6}	D14	{A1,A4,A6}
AD15	D15,{A2}	D15,{A2}	D15	{A2}
AD16	D16,{A4,A5,A6}	D16,{A4,A5,A6}	D16	{A4,A5,A6}

AD17	D17,{A3,A5}	D17,{A3,A5}	D17	{A3,A5}
AD18	D18,{A4}	D18,{A4}	D18	{A4}
AD19	D19,{A5,A6}	D19,{A5,A6}	D19	{A5,A6}

În general pentru ca o clasă să poată fi derivabilă este necesar ca aceasta să conțină informații din clasa de bază. În procesul implementării de multe ori este posibil ca utilizatorii să ceară introducerea de informație nouă. Aceasta poate fi introdusă sub forma unor clase nederivabile sau prin modificarea claselor derivabile existente. Modificarea claselor existente, poate avea implicații destul de neplăcute asupra programelor și asupra informațiilor care vor ajunge la utilizator.

8.4.2 Definirea schemele conceptuale și a schemelor externe

Așa cum s-a arătat anterior schema conceptuală a unei baze de date obiectuală conține clase nederivabile. Schema obiectuală va fi generată pe baza metodologiei prezentată în cap. 4 paragraf 4.3 și pe baza algoritmului prezentat în cap. 5, paragraf 5.9.2. Ea poate conține și clase derivabile dar acest lucru este opțional. Informația conținută în schema obiectuală în general este reprezentată prin proprietățile clasei care reprezintă *intensia* clasei și prin mulțimea obiectelor care există într-o clasă care reprezintă *extensia* clasei. Intensia și extensia sunt considerate în general separat. Astfel dacă se include un obiect nou într-o clasa derivată același obiect trebuie inclus și în clasa de bază. La fel modificarea unor proprietăți în clasa de bază implică modificarea acestora și în clasa derivată. Clasele derivate pot conține informație nederivabilă.

Schemele externe derivă din schema conceptuală, prin introducerea de noi clase derivate. Ele descriu o parte din informația existentă în baza de date în scopul de a transmite informație utilizatorului. Așa cum s-a arătat în capitolele anterioare schemele externe pot conține informație care a fost obținută prin semantica obiect generat. Gradul de particularizare a schemei externe poate merge până a reprezenta o sungură clasă.

Pornind de la aceste considerații teoretice, schema conceptuală a aplicației conține clasele nederivabile, necesare pentru memorarea datelor, pentru butoane, pentru gestionarea datelor, pentru obținerea principalelor documente economice, pentru realizarea părții de proiectare, pentru vizualizarea STAS-urilor, pentru gestiunea producției (figura 8.22).

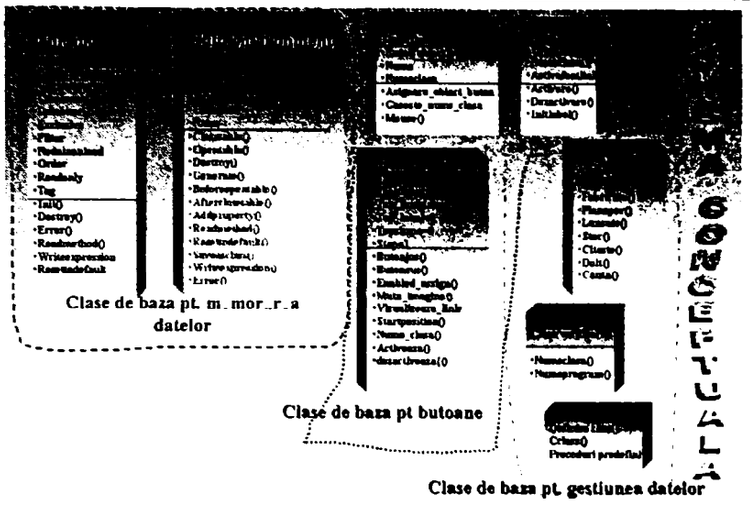


Fig 8.22 Clase de bază

Clasele pentru manipularea datelor sau diverse secțiuni ale aplicației care deservesc Sistemul de Fabricație al Sculelor, se bazează pe clase predefinite ale me iului de programare (figura 8.23).

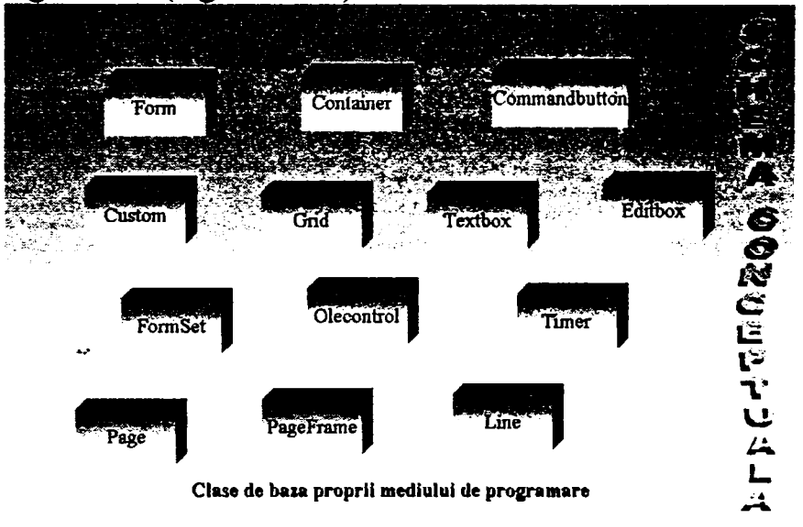


Fig 8.23 Clase predefinite ale mediului de programare

Schemele externe sunt specifice fiecărei secțiuni a bazei de date, ele se bazează pe schemea conceptuală dar conțin şi alte clase (clase derivabile) decât clasele de bază. Instanțele clasei de origine pot deveni instanțe ale clasei destinație în anumite condiții precis definite. Procesul este complex si generează în procesul de proiectare noi clase derivabile sau nederivabile care apar în schema externă. Schema conceptuală poate fi modificată pe parcurs în scopul de a obține clase nederivabile. Într-o primă fază pot fi generate scheme externe temporale care se modifică continu până se ajunge la schema externă corespunzătoare unei părți din baza de date obiectuală. În general prin modificarea schemei conceptuale, schemele externe inițiale ar trebui să nu se modifice.

8.4.2.1. Schema externă a bazei de date obiectuale "Alezoare"

Pornind de la considerentele prezentate anterior, s-a folosit un produs soft de modelare Visual Modeler cu ajutorul căruia s-au obținut schemele obiectuale pentru fiecare modul al SFS-ului conform clasificării efectuate pentru sculele așchietoare. Schema externă pentru secțiunea baza de date "alezoare" este prezentată în figura 8.24. Într-un modul al bazei de date obiectuale a sistemului de fabricație al sculelor pot exista mai multe scheme obiectuale fiecare reprezentând structura de clase dorită. În cazul bazei de date alezoare este inclusă și schema externă corespunzătoare "Planului de Operații" (figura 8.25). Totodată clasele folosite pentru generarea documentelor economice sau cele folosite pentru partea de proiectare sunt clase derivate, în general prin moștenire care au propriile lor scheme externe.

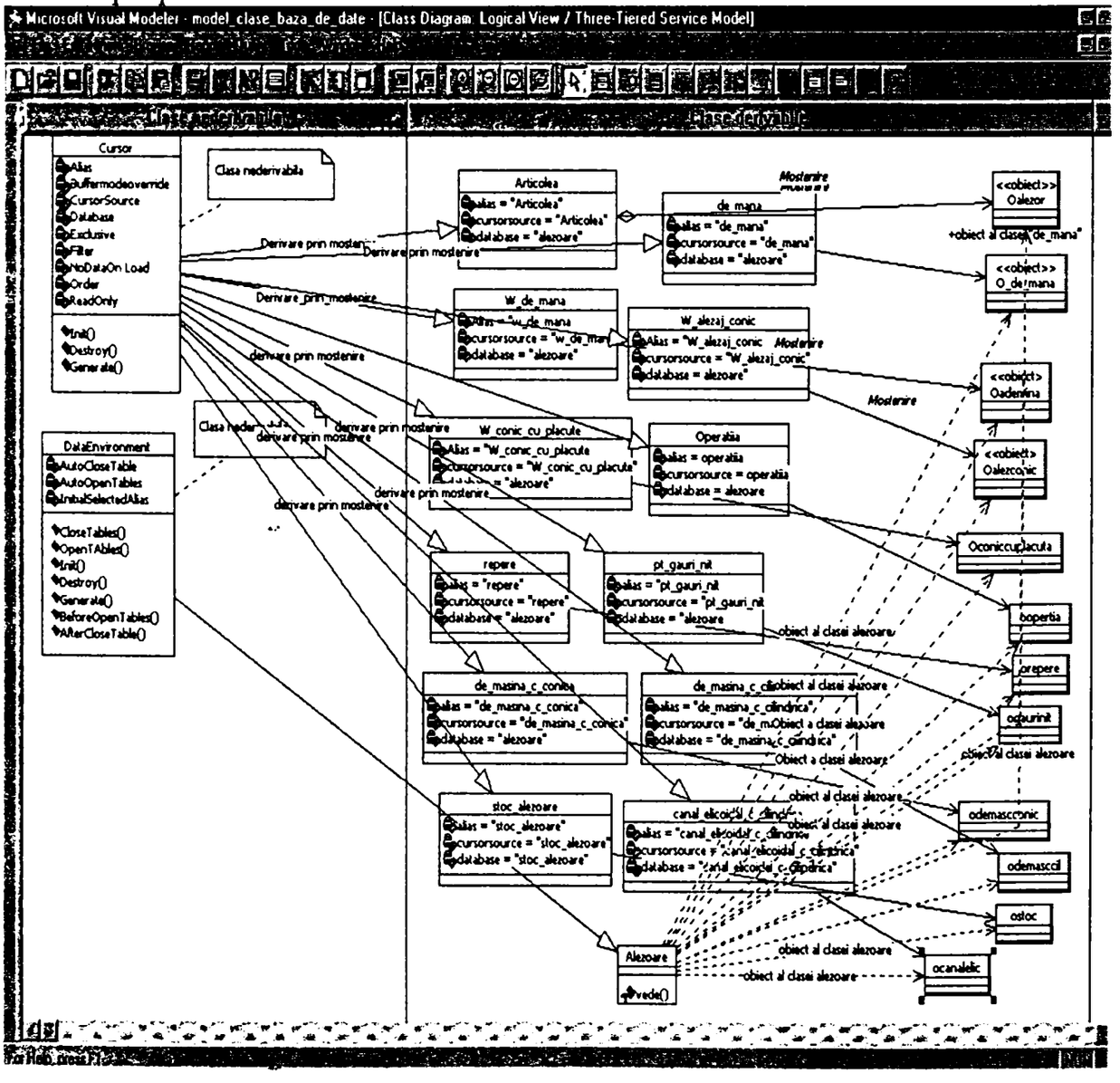


Fig 8.24 Schema externă "Alezoare"

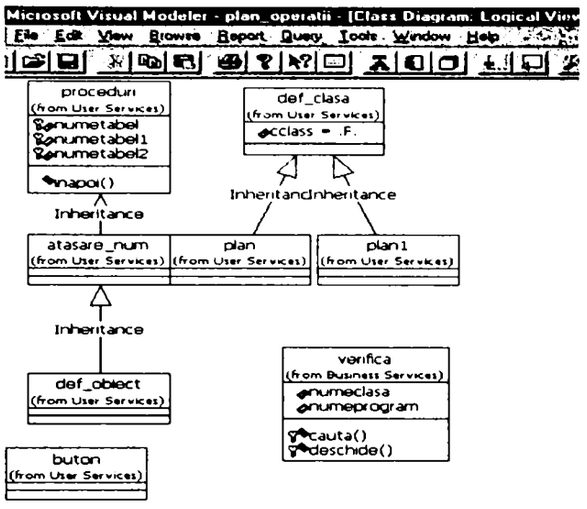


Fig 8.25 Schema externă "Plan operații"

8.4.2.2. Schema externă a bazei de date obiectuale "Burghie"

Schema externă pentru clasele care generează baza de date obiectuală "Burghie" este prezentată în figura 8.26.

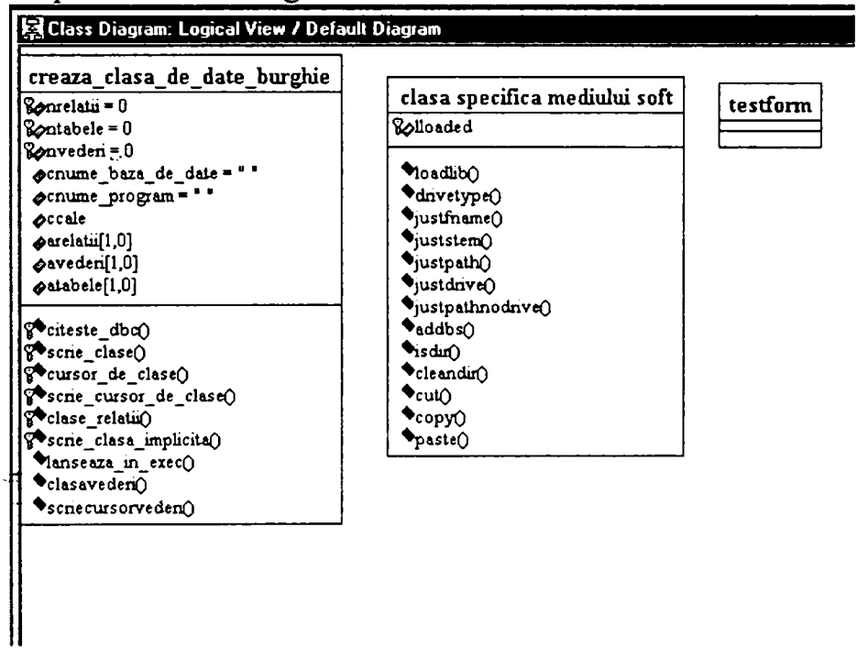


Fig 8.26 Schema externă a claselor care generează baza de date obiectuală "Burghie"

Această schemă obiectuală este formată din clase derivate, din clasele de bază, adaptate pentru burghie. Demn de remarcat este faptul că o bază de date obiectuală este generată cu ajutorul unor clase derivate, derivarea realizându-se prin moştenire, cu ajutorul semanticii obiect generat. Se porneşte de la o bază de date relaţională şi printr-o tratare pur obiectuală se ajunge la o bază de date obiectuală.

Schema obiectuală a bazei de date obiectuale burghie este prezentată în figura 8.27. Baza de date obiectuală este formată din clase derivate din clasele de bază *cursor*. Ea este tot o clasă derivată de tip dataenvironment care conține un număr de obiecte. Numărul de obiecte este variabil el putând fi modificat atât în procesul de implementare, cât și în timpul întreținerii bazei de date. Modul în care au fost generate bazele de date, schemele externe, respectă strict teoria enunțată pe parcursul tezei. Modul în care a fost implementată baza de date, atât din punct de vedere, teoretic, al proiectării cât și al generării obiectual peste relațional în sistemul de fabricație al sculelor constitue o noutate.

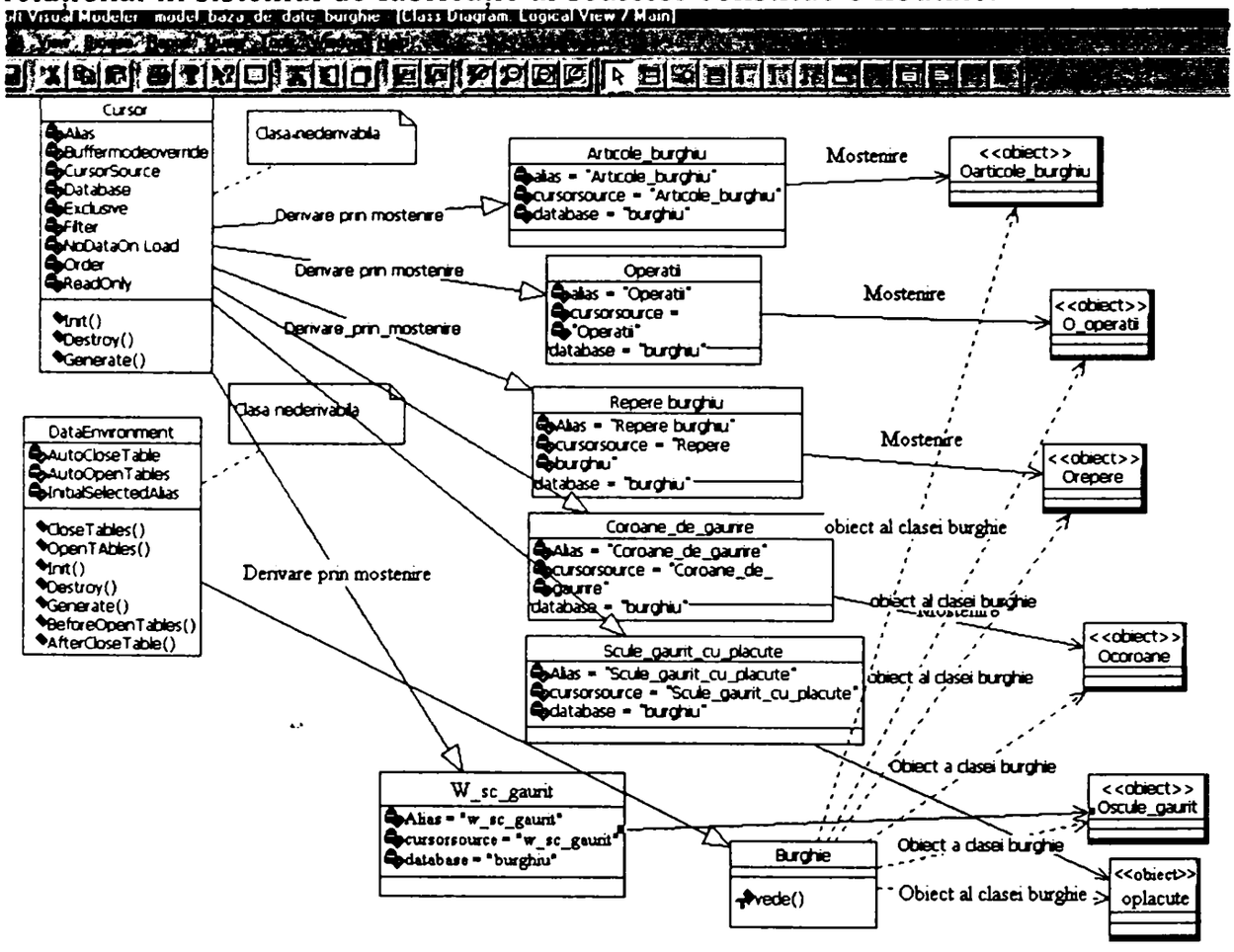


Fig 8.27 Schema obiectuală a bazei de date "Burghie"

Schema obiectuală pentru gestionarea informațiilor din baza de date burghie este prezentată în figura 8.28. Clasele care aparțin acestei scheme obiectuale, derivă din clasele de bază cu referire strictă la gestionarea și vizualizarea informațiilor. Operațiile fundamentale dintr-o bază de date, introducere, căutarea, modificarea ștergerea și vizualizarea sunt realizate prin clase de obiecte, clase derivabile prin semantica obiect generat clase care obțin informația stocată în obiecte ale clasei "Burghie".

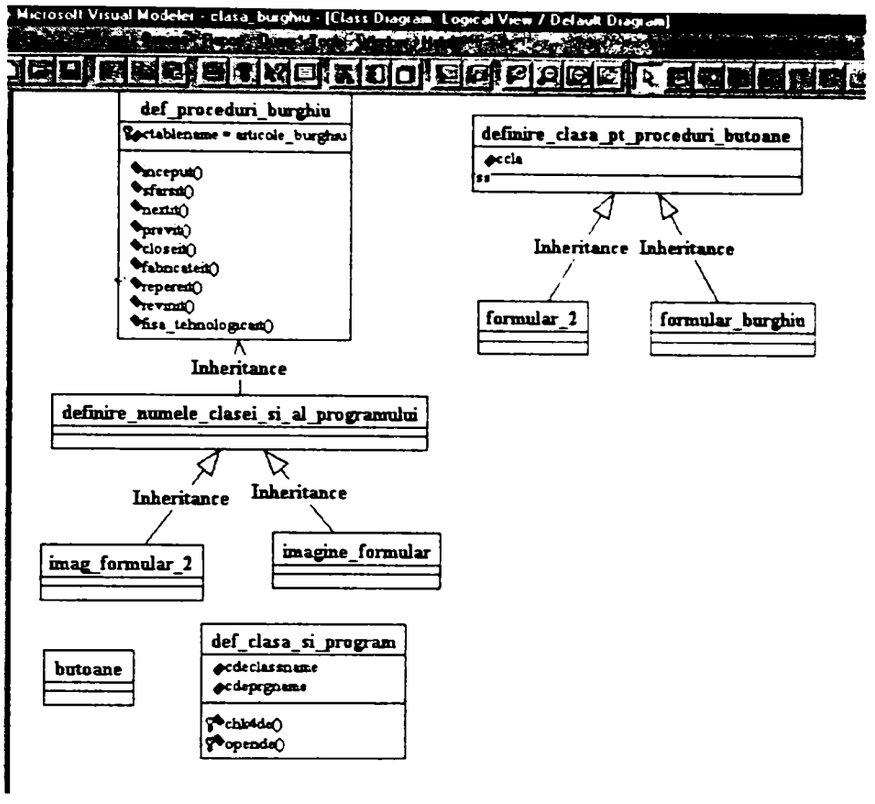


Fig 8.28 Schema obiectuală pentru gestionarea informațiilor din baza de date "Burghie"

8.4.2.2. Schema externă a bazei de date obiectuale "Cutite"

În mod asemănător, au fost obținute clasele pentru modulul "Cuțite" din cadrul bazei de date. Schema obiectuală pentru clasele care formează baza de date, este prezentată în figura 8.29.

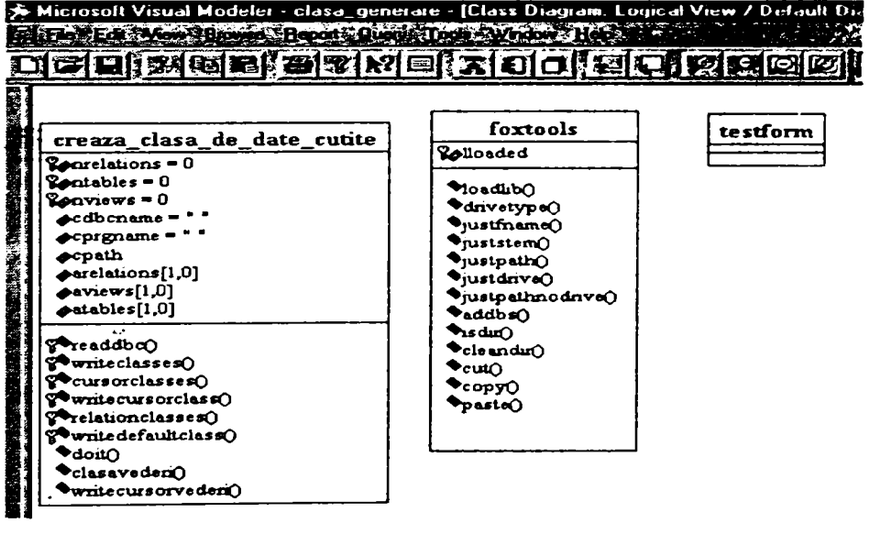


Fig 8.29 Schema obiectuală a claselor pentru crearea claselor de memorare a bazei de date obiectuale "Cutite"

Schema obiectuală a bazei de date obiectuale "Cuțite" este prezentată în figura 8.30.

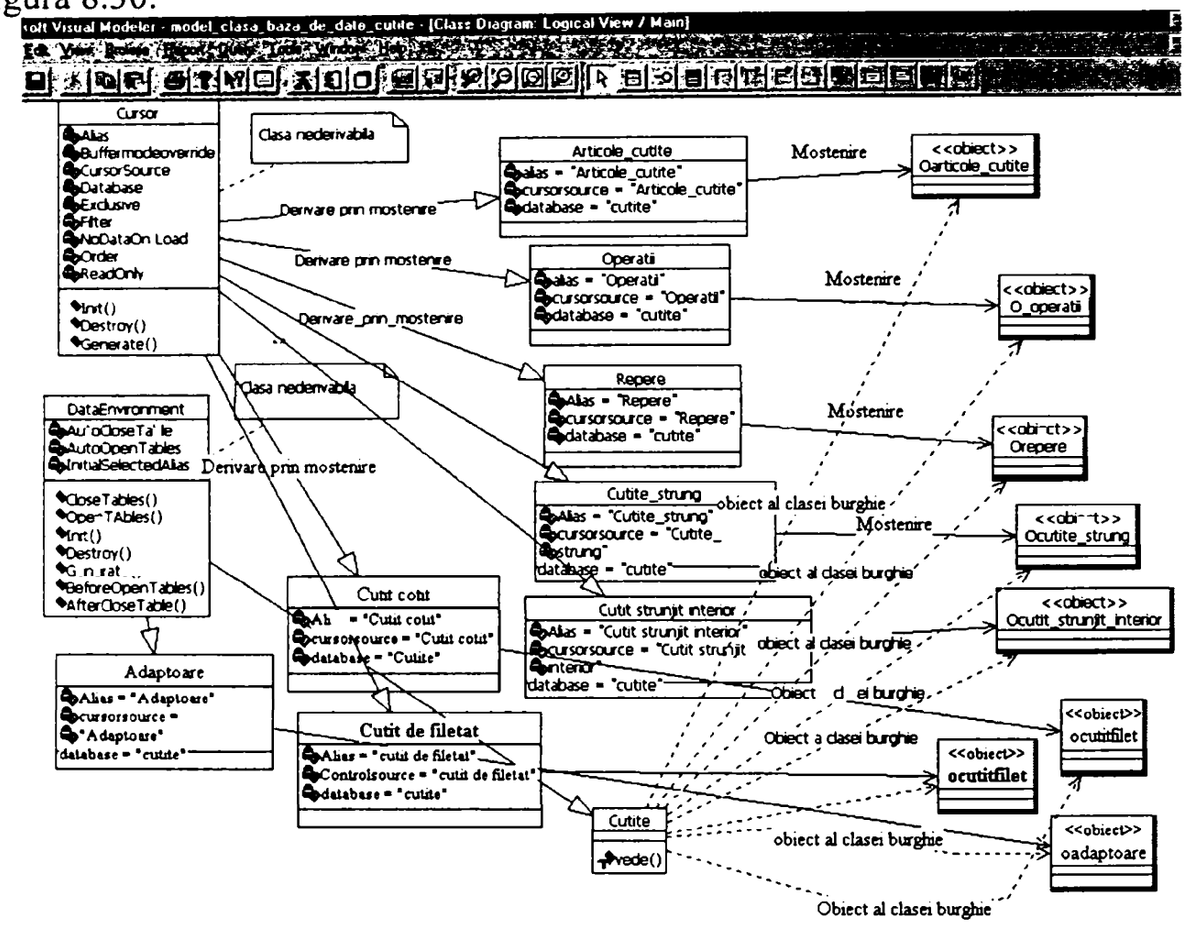


Fig 8.30 Schema obiectuală a bazei de date obiectuale "Cuțite"

Schema externă pentru clasele care gestionează datele din baza de date "Cuțite" este prezentată în figura 8.31.

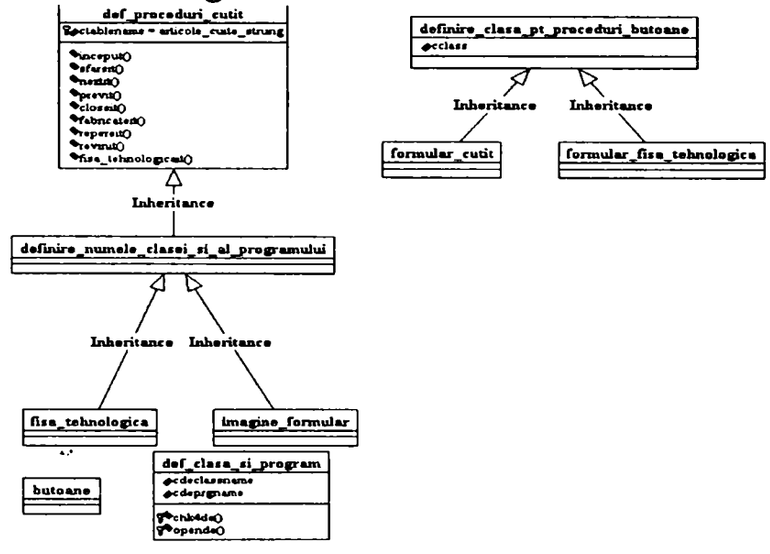


Fig 8.31 Schema externă pentru clasele care gestionează datele din baza de date "Cuțite"

8.4.2.3. Schemele externe a bazei de date obiectuale "Freze"

În mod asemănător, au fost obținute clasele pentru modulul "Freze" din cadrul bazei de date a Sistemului de Fabricație al Sculelor. Schema obiectuală, pentru clasele care crează baza de date, este prezentată în figura 8.32.

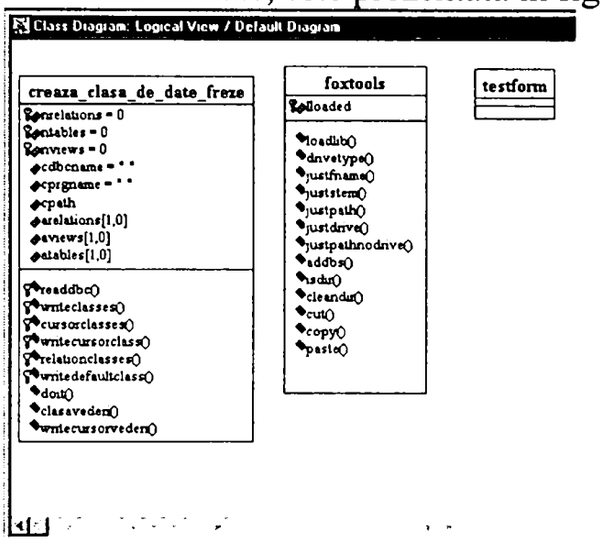


Fig 8.32 Schema obiectuală, pentru clasele care crează baza de date "Freze"

Schema externă a bazei de date obiectuale "Freze" este prezentată în figura 8.33.

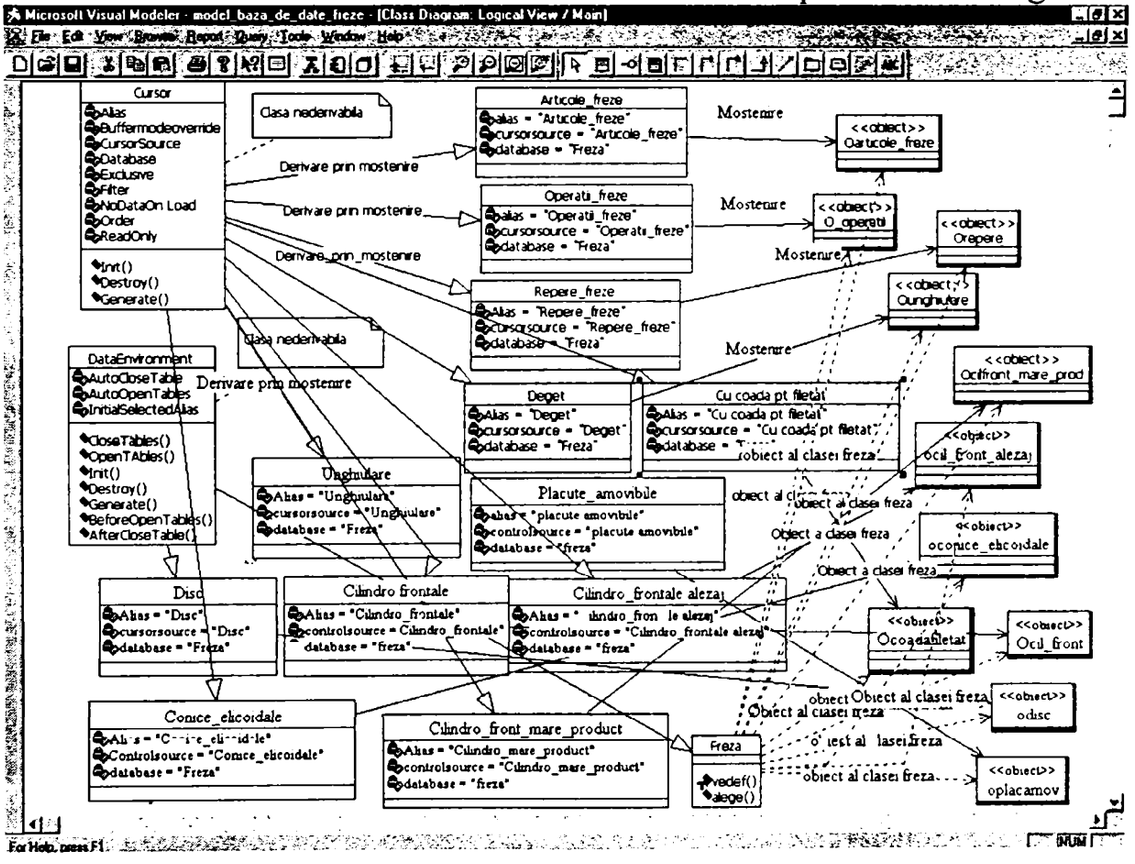


Fig 8.33 Schema externă a bazei de date obiectuale "Freze"

Schema externă a claselor care gestionează datele din baza de date "Freze" este prezentată în figura 8.34.

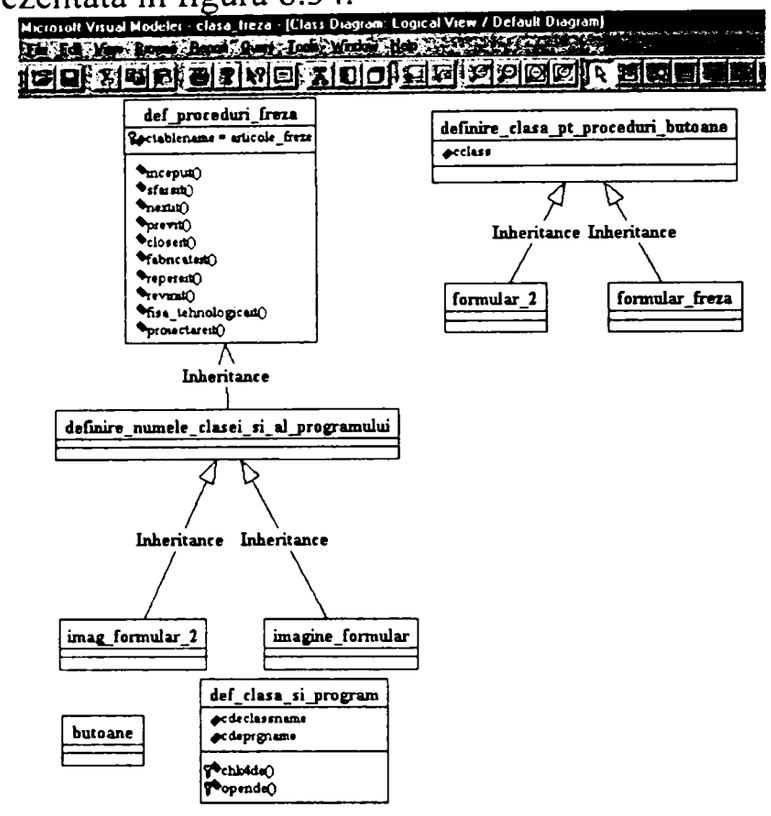


Fig 34 Schema externă a claselor care gestionează datele din baza de date "Freze"

8.4.2.4.1 Pregătirea fabricației pentru baza de date obiectuală "Freze"

În general pregătirea fabricației reprezintă unul dintre punctele critice ale fluxului informațional, care de regulă consumă foarte mult timp şi încetineşte apariția produselor noi pe piață. Nu cu mult timp în urmă pregătirea fabricației consuma un timp îndelungat, de la câteva săptămâni la câteva luni, iar colaborat cu alte puncte critice întârzia apariția produselor pe piața, la ordinul anilor, moment în care produsul era uzat moral. Nu întâmplător primele țări care au făcut progrese în rezolvarea acestor probleme, ajungând la soluții deosebite sunt S.U.A, Japonia, țări în care sistemul informațional a fost analizat, dezvoltat şi optimizat, înaintea altora. Acest studiu urmăreşte parcurgerea rapidă a unor paşi clasici, prin care în cadrul proiectării procesului tehnologic a unei piese se urmăreşte indicarea unei scule adecvate pentru anumite operații, sculă care nu este proiectată până la momentul respectiv şi bineînțeles nici fabricată.

Proiectarea sculei şi elaborarea procesului tehnologic de fabricație al acestora ar cere destul de mult timp şi ar consuma activități necesare procesului de corelare şi transmitere a documentelor respective (pentru varianta clasică de proiectare şi circulație a informației).

Pregătirea urmăreşte proiectarea unei scule reale plecând de la modele abstracte, modelul generalizat şi scula reprezentativă. În figura 8.35 este

prezentată clasa "Freză generalizată". Scula generalizată, este o sculă abstractă, fără posibilități de aşchiere, dar cu posibilități de vizualizare a sculelor existente, caracterizate prin diverse tipuri de suprafețe (frontale, cilindrice, conice, profilate, etc.), prin număr diferit de scule aşchietoare, cu poziții diferite de aşezare a acestora (tangențial, radial) şi cu diferite posibilități de mişcare (rectilinii sau de rotație).

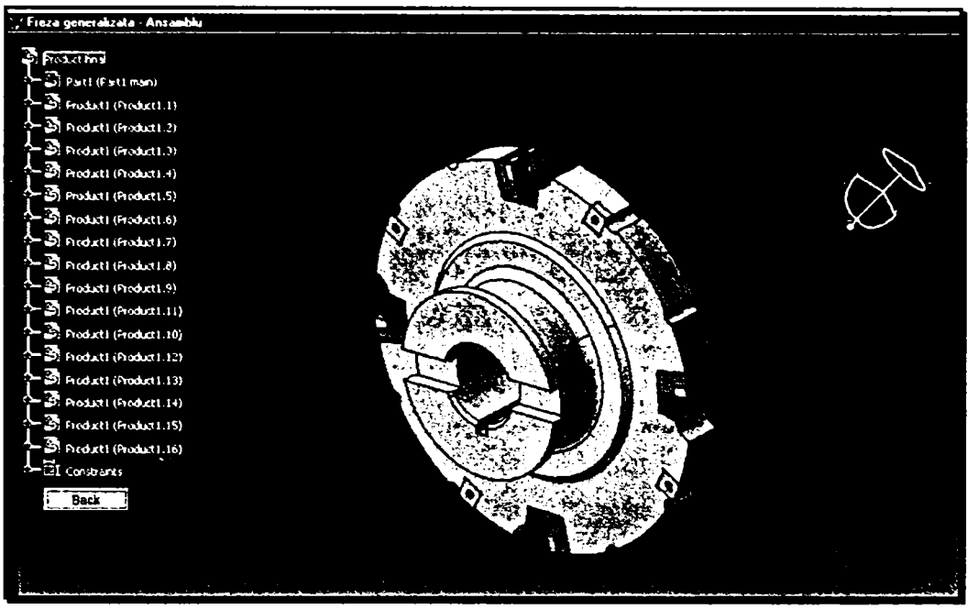


Fig 8.35 Clasa "Freza generalizată"

Prin particularizarea sculei generalizate, se poate ajunge la diferite tipuri de scule reprezentative, pe categorii de prelucrări (frezare frontală, frezare plană, frezare laterală, frezare cilindro frontală, etc.). În general proiectarea unei scule reprezentative se face pornind de la identificarea elementelor comune tuturor particularizărilor acesteia şi adăugarea elementelor specifice, care diferențiază sculele reale ce pot fi obținute. Se ajunge la imaginea sculei reale, clasa "Freza reprezentativă" (figura 8.36). Modelul sculei reprezentative cuprinde toate celelalte scule cu care se poate realiza operația respectivă. Desigur modelul generalizat conține toate tipurile de scule şi de la el se ajunge la scula reprezentativă pe operație.

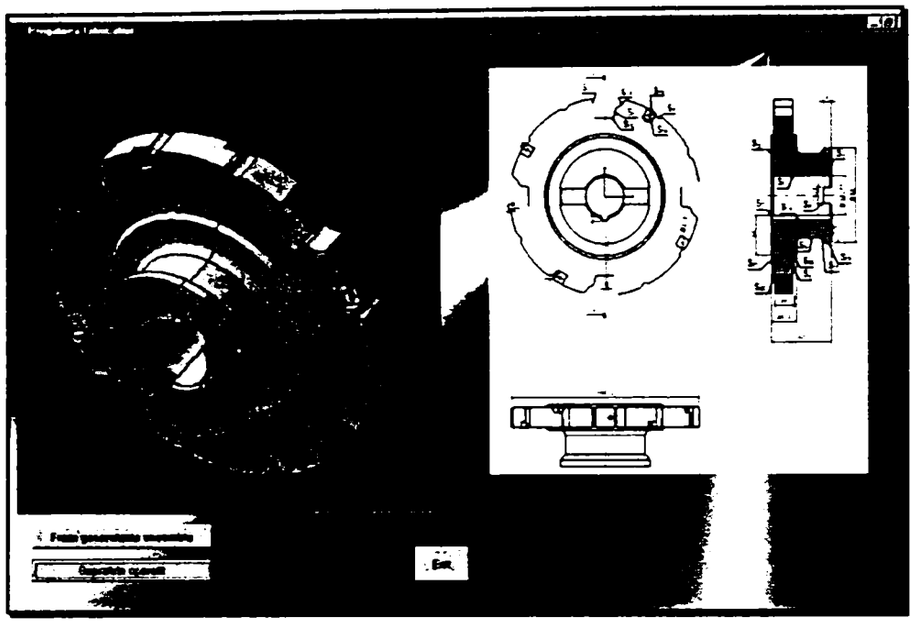


Fig 8.36 Clasa "Freza reprezentativă"

În aplicația realizată care are ca suport de memorare al datelor o bază de date obiectual relațională, iar gestionarea informației se realizează pur obiectual, se pleacă de la o sculă reprezentativă, freză disc pentru frezarea canalelor şi se ajunge la diferite tipuri de freze disc, cu plăcuțe schimbabile aşezate radial sau tangențial. Freza reprezentativă din categoria freze disc este caracterizată de un număr de suprafețe şi operații aferente acestora. Această freză constitue un punct de plecare pentru constituirea unei freze reale, cu posibilități de aşchiere, prin simpla îndepărtare a suprafețelor care nu îndeplinesc condițiile cerute de un anume tip de freză. În figurile 8.37 şi 8.38, sunt prezentate clasele "Freză disc cu plăcuțe aşezate radial" şi "Freză disc cu plăcuțe aşezate tangențial".

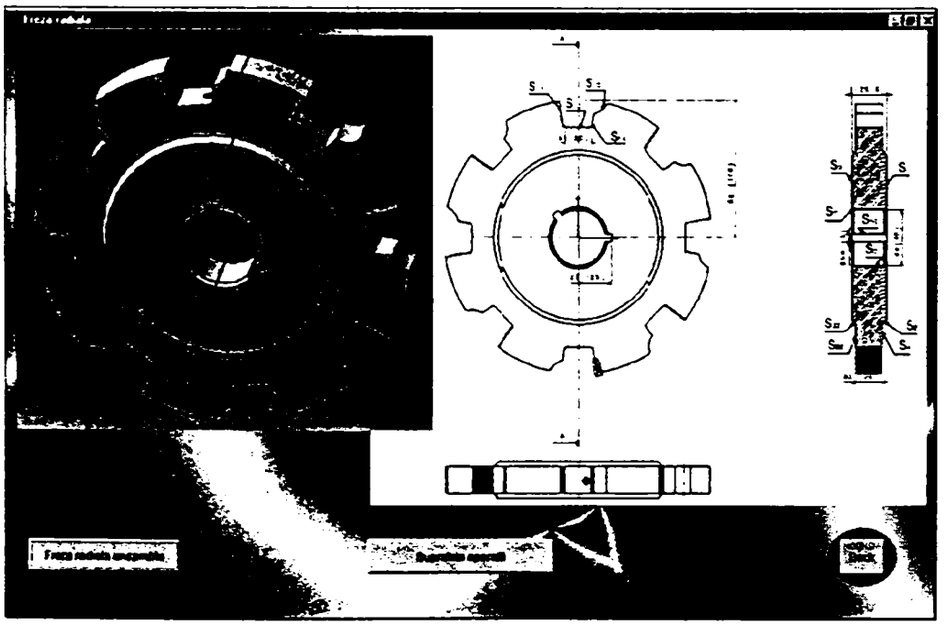


Fig 8.37 Clasa "Freză disc cu plăcuțe aşezate radial"

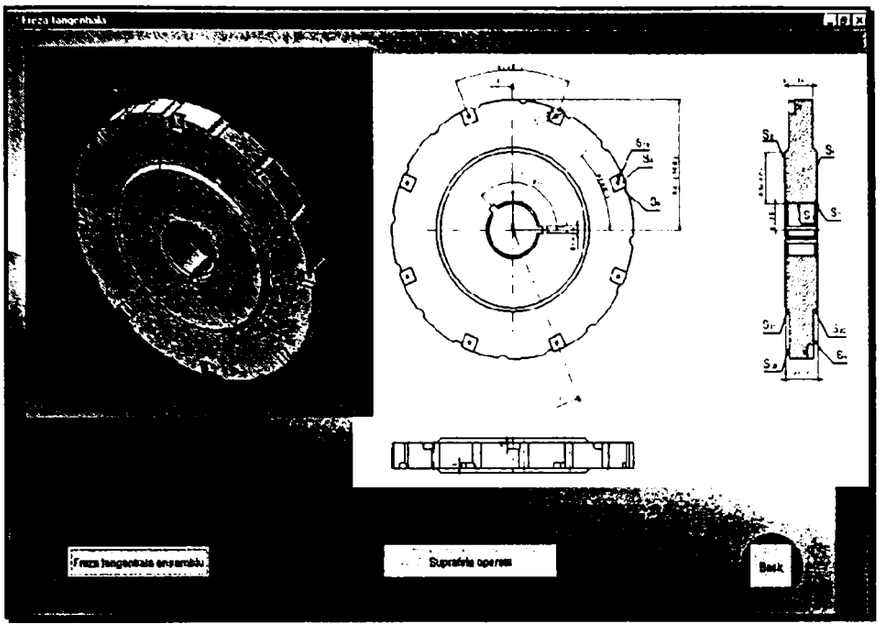


Fig 8.38 Clasa"Freză disc cu plăcuțe așezate tangențial"

Clasa "Freza disc cu plăcuțe așezate radial de ansamblu" este prezentată în figura 8.39. Imaginile prezentate sunt din cadrul aplicației, butoanele aferente permițând accesul la diverse obiecte care reprezintă informația dorită.

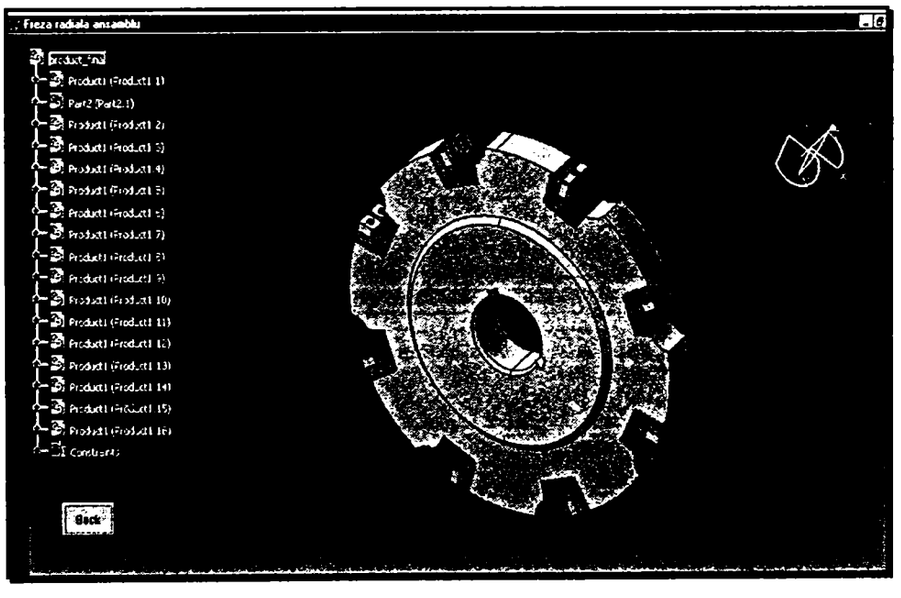


Fig 8.39 Clasa "Freza disc cu plăcuțe așezate radial de ansamblu"

Clasa "Freza disc cuplăcuțe așezate tangențial de ansamblu" este prezentată în figura 8.40.

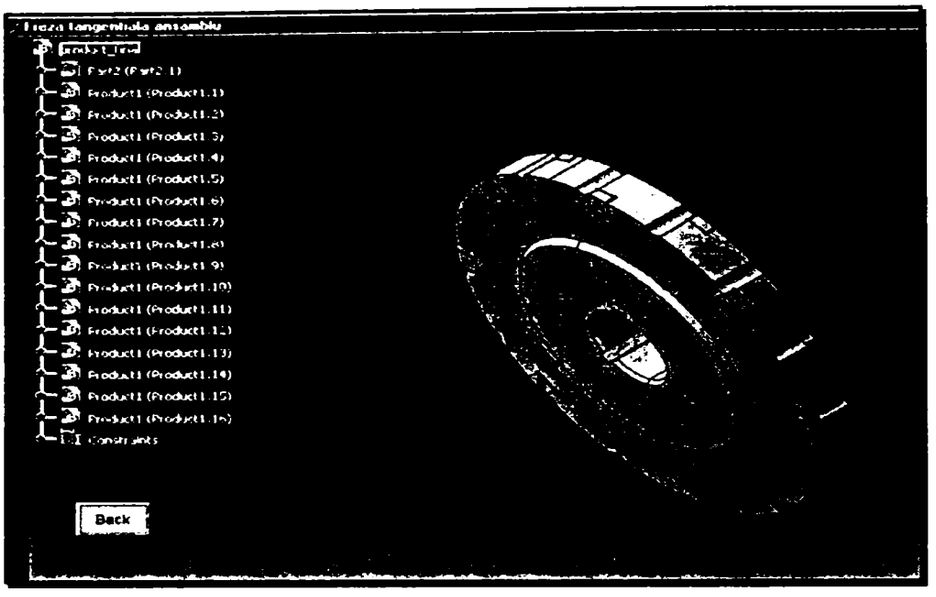


Fig 8.40 Clasa "Freza disc cuplăcuțe așezate tangențial de ansamblu"

În baza de date obiectuală, există proiectată tehnologia, pentru scula reprezentativă, clasa "Suprafețe pentru freza disc reprezentativă"(figura 8.41) şi prin particularizare, se elimină operațiile pentru suprafețele pe care nu le conține scula aleasă (reală). Astfel se ajunge la tehnologia de fabricație a pentru freza disc cu placuțe aşezate radial, clasa "Suprafețe pentru freza disc cu plăcuțe aşezate radial" (figurile 8.42) şi clasa "Suprafețe pentru freza disc cu plăcuțe aşezate tangențial" (figura 8.43).

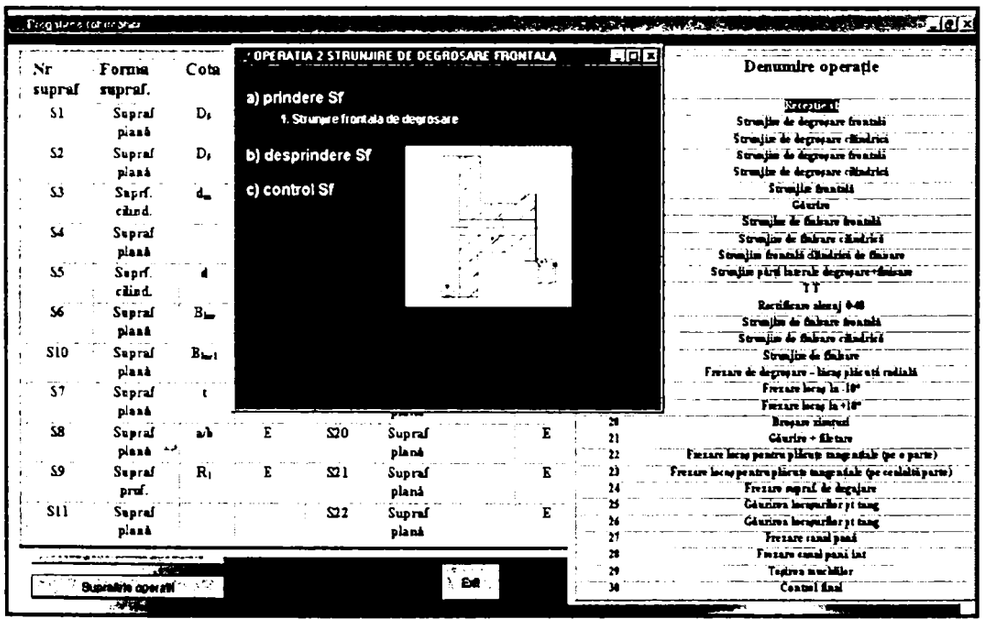
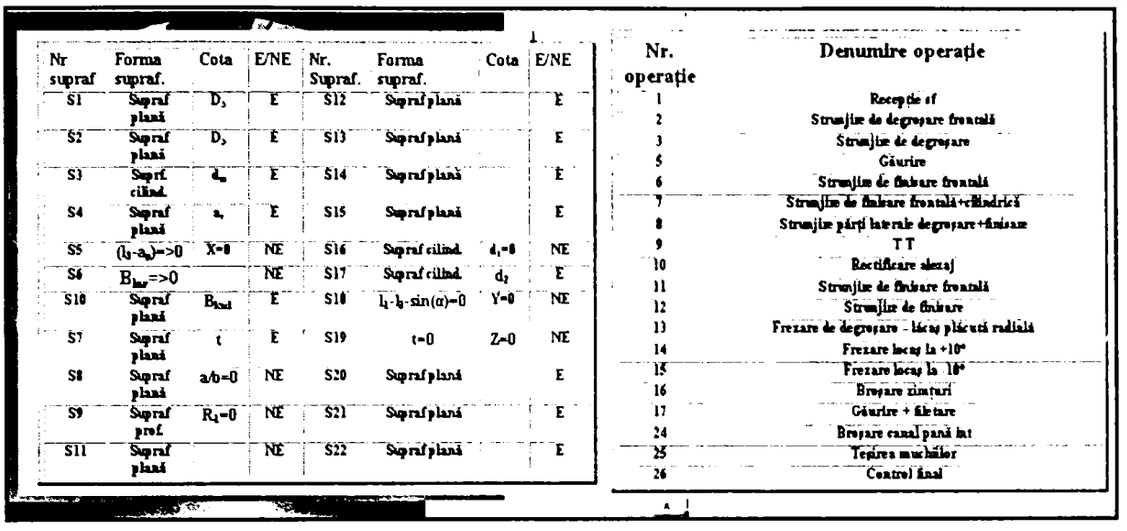


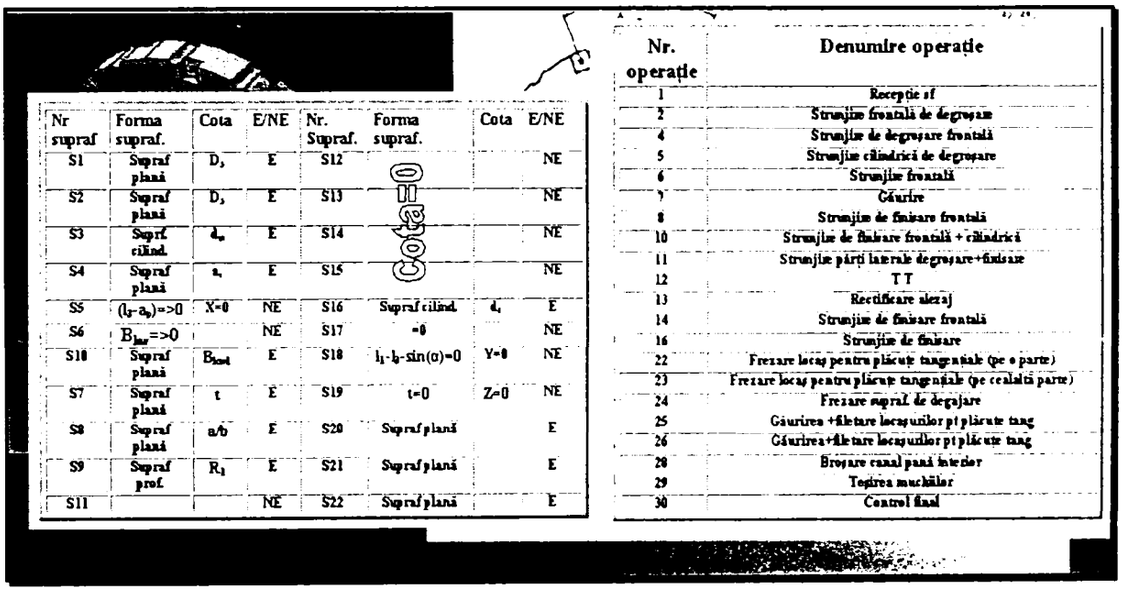
Fig 8.41 Clasa "Suprafețe pentru freza disc reprezentativă"



Nr supraf	Forma supraf.	Cota	E/NE	Nr. Supraf.	Forma supraf.	Cota	E/NE
S1	Suprafață plană	D_3	E	S12	Suprafață plană		E
S2	Suprafață plană	D_3	E	S13	Suprafață plană		E
S3	Suprf. cilind.	d_m	E	S14	Suprafață plană		E
S4	Suprafață plană	a_1	E	S15	Suprafață plană		E
S5	$(l_3-a_3)=>0$	X=0	NE	S16	Suprafață cilind.	$d_1=0$	NE
S6	$B_{bar}=>0$		NE	S17	Suprafață cilind.	d_2	E
S10	Suprafață plană	B_{bar}	E	S18	$l_1-l_3-\sin(\alpha)=0$	Y=0	NE
S7	Suprafață plană	t	E	S19	t=0	Z=0	NE
S8	Suprafață plană	a/b=0	NE	S20	Suprafață plană		E
S9	Suprafață prof.	$R_1=0$	NE	S21	Suprafață plană		E
S11	Suprafață plană		NE	S22	Suprafață plană		E

Nr. operație	Denumire operație
1	Recepție sf
2	Strunjire de degroșare frontală
3	Strunjire de degroșare
5	Găurire
6	Strunjire de finisare frontală
7	Strunjire de finisare frontală+cilindrică
8	Strunjire părți laterale degroșare+finisare
9	T T
10	Rectificare alezaj
11	Strunjire de finisare frontală
12	Strunjire de finisare
13	Frezare de degroșare - lăcaș plăcuță radială
14	Frezare locaș la +10°
15	Frezare locaș la -10°
16	Broșare canaturi
17	Găurire + filetare
24	Broșare canal pană int
25	Tesirea muchiilor
26	Control final

Fig 8.42 Clasa "Suprafețe pentru freza disc cu plăcuțe așezate radial"



Nr supraf	Forma supraf.	Cota	E/NE	Nr. Supraf.	Forma supraf.	Cota	E/NE
S1	Suprafață plană	D_3	E	S12	Cotă 0		NE
S2	Suprafață plană	D_3	E	S13			NE
S3	Suprf. cilind.	d_m	E	S14			NE
S4	Suprafață plană	a_1	E	S15			NE
S5	$(l_3-a_3)=>0$	X=0	NE	S16	Suprafață cilind.	d_1	E
S6	$B_{bar}=>0$		NE	S17	=0		NE
S10	Suprafață plană	B_{bar}	E	S18	$l_1-l_3-\sin(\alpha)=0$	Y=0	NE
S7	Suprafață plană	t	E	S19	t=0	Z=0	NE
S8	Suprafață plană	a/b	E	S20	Suprafață plană		E
S9	Suprafață prof.	R_1	E	S21	Suprafață plană		E
S11			NE	S22	Suprafață plană		E

Nr. operație	Denumire operație
1	Recepție sf
2	Strunjire frontală de degroșare
4	Strunjire de degroșare frontală
5	Strunjire cilindrică de degroșare
6	Strunjire frontală
7	Găurire
8	Strunjire de finisare frontală
10	Strunjire de finisare frontală + cilindrică
11	Strunjire părți laterale degroșare+finisare
12	T T
13	Rectificare alezaj
14	Strunjire de finisare frontală
16	Strunjire de finisare
22	Frezare locaș pentru plăcuțe tangențiale (pe o parte)
23	Frezare locaș pentru plăcuțe tangențiale (pe cealaltă parte)
24	Frezare suprafață de degajare
25	Găurirea +filetare locașurilor pt plăcuțe tang
26	Găurirea+filetare locașurilor pt plăcuțe tang
28	Broșare canal pană interior
29	Tesirea muchiilor
30	Control final

Fig 8.43 Clasă "Suprafețe pentru freza disc cu plăcuțe așezate tangențial"

Această ramură a aplicației poate fi consultată la adresa www.blanka.org. Codul sursă a claselor pentru această ramură este următorul:

```
DEFINE CLASS piesa_generalizata AS formset
DataSession = 1
AutoRelease = .T.
Name = "piesa_generalizata"
ADD OBJECT form1 AS form WITH ;
        Height = 742, ;
        Width = 1017, ;
        ShowWindow = 0, ;
        DoCreate = .T., ;
        AutoCenter = .T., ;
        Caption = "Examen de diploma", ;
        Closable = .F., ;
        MaxButton = .F., ;
        MinButton = .F., ;
        Movable = .F., ;
        WindowState = 2, ;
        BackColor = RGB(0,0,0), ;
        Name = "Form1"
ADD OBJECT piesa_generalizata.form1.command1 AS commandbutton WITH ;
        Top = 528, ;
        Left = 24, ;
        Height = 25, ;
        Width = 204, ;
        Caption = "Freza generalizata ansamblu", ;
        Name = "Command1"
ADD OBJECT piesa_generalizata.form1.image1 AS image WITH ;
        Picture = "freza_generalizata_2.jpg", ;
        Stretch = 1, ;
        BackStyle = 0, ;
                Height = 504, ;
                Left = 0, ;
                Top = 12, ;
                Width = 492, ;
                Name = "Image1"
```

```
ADD OBJECT piesa_generalizata.form1.image2 AS image
WITH .
                Picture                =
"freza_generalizata_executie.jpg", ;
                Stretch = 0, ;
                Height = 439, ;
                Left = 540, ;
                Top = 36, ;
                Width = 423, ;
                Name = "Image2"
ADD OBJECT piesa_generalizata.form1.commandgroup1 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 60, ;
                Left = 120, ;
                Top = 96, ;
                Width = 60, ;
                Name = "Commandgroup1", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup2 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 60, ;
                Left = 348, ;
                Top = 240, ;
                Width = 60, ;
                Name = "Commandgroup2", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup3 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 60, ;
                Left = 264, ;
                Top = 420, ;
                Width = 60, ;
                Name = "Commandgroup3", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup4 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 60, ;
                Left = 36, ;
                Top = 276, ;
                Width = 60, ;
                Name = "Commandgroup4", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup5 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 60, ;
                Left = 48, ;
                Top = 132, ;
                Width = 60, ;
                Name = "Commandgroup5", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.image3 AS image
WITH ;
                Picture = "tabel_generalizat.jpg", ;
                Stretch = 2, ;
                Height = 532, ;
                Left = 0, ;
                Top = 12, ;
                Visible = .F., ;
                Width = 612, ;
                Name = "Image3"
ADD OBJECT piesa_generalizata.form1.image4 AS image
WITH ;
                Picture                =
"tabel_operatii_generalizate.jpg", ;
                Stretch = 2, ;
                Height = 581, ;
                Left = 588, ;
                Top = 12, ;
                Visible = .F., ;
                Width = 435, ;
                Name = "Image4"
ADD OBJECT piesa_generalizata.form1.command2 AS
commandbutton WITH ;
                Top = 564, ;
                Left = 24, ;
                Height = 25, ;
                Width = 204, ;
                Caption = "Suprafete operatii", ;
                Name = "Command2"
ADD OBJECT piesa_generalizata.form1.commandgroup6 AS
commandgroup WITH ;
```

```
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 16, ;
                Left = 732, ;
                Top = 84, ;
                Width = 205, ;
                Visible = .F., ;
                BorderColor = RGB(255,255,255), ;
                Name = "Commandgroup6", ;
                Command1.Top = 0, ;
                Command1.Left = -24, ;
                Command1.Height = 32, ;
                Command1.Width = 12, ;
                Command1.Caption = "", ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup7 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 12, ;
                Left = 732, ;
                Top = 102, ;
                Width = 204, ;
                Visible = .F., ;
                BorderColor = RGB(255,255,255), ;
                Name = "Commandgroup7", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 6, ;
                Command1.Left = -24, ;
                Command1.Height = 26, ;
                Command1.Width = 18, ;
                Command1.Caption = "", ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup8 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 12, ;
                Left = 744, ;
                Top = 120, ;
                Width = 204, ;
                Visible = .F., ;
                BorderColor = RGB(255,255,255), ;
                Name = "Commandgroup8", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 6, ;
                Command1.Left = -24, ;
                Command1.Height = 26, ;
                Command1.Width = 18, ;
                Command1.Caption = "", ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup9 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 72, ;
                Left = 288, ;
                Top = 156, ;
                Width = 84, ;
                Name = "Commandgroup9", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.commandgroup10 AS
commandgroup WITH ;
                AutoSize = .F., ;
                ButtonCount = 1, ;
                BackStyle = 0, ;
                BorderStyle = 0, ;
                Value = 1, ;
                Height = 72, ;
                Left = 336, ;
                Top = 396, ;
                Width = 60, ;
                Name = "Commandgroup10", ;
                Command1.AutoSize = .F., ;
                Command1.Top = 5, ;
                Command1.Left = 5, ;
                Command1.Height = 27, ;
                Command1.Width = 84, ;
                Command1.Caption = "", ;
                Command1.Visible = .F., ;
                Command1.ColorScheme = 1, ;
                Command1.Name = "Command1"
ADD OBJECT piesa_generalizata.form1.command3 AS
commandbutton WITH ;
                Top = 552, ;
                Left = 456, ;
                Height = 37, ;
                Width = 61, ;
                Caption = "Exit", ;
                Name = "Command3"


ADD OBJECT form2 AS form WITH ;
        Top = 0, ;
        Left = 0, ;
        Height = 742, ;
        Width = 1017, ;
        DoCreate = .T., ;
Caption = "FREZA GENERALIZATA ANSAMBLU", ;
                Visible = .F., ;
                WindowState = 2, ;
                BackColor = RGB(0,0,0), ;
                Name = "Form2"
ADD OBJECT piesa_generalizata.form2.image1 AS image
WITH ;
        Picture = "freza_generalizata_ansamblu1.jpg", ;
                Height = 619, ;
                Left = 12, ;
                Top = 12, ;
                Width = 984, ;
                Name = "Image1"
ADD OBJECT piesa_generalizata.form2.command1 AS
commandbutton WITH ;
                Top = 444, ;
                Left = 36, ;
                Height = 25, ;
                Width = 85, ;
                Caption = "Back", ;
                Visible = .F., ;
                Name = "Command1"
        ADD OBJECT form3 AS form WITH ;
                Top = 0, ;
                Left = 0, ;
                Height = 742, ;
                Width = 1017, ;
```

```
                    DoCreate = .T., ;
                    Caption = "Freza tangentiala", ;
                    WindowState = 2, ;
                    BackColor = RGB(0,0,0), ;
                    Name = "Form3"
ADD OBJECT piesa_generalizata.form3.image2 AS image
WITH ,
Picture = "freza_tangentiala_executie_1.jpg", ;
                    Stretch = 1, ;
                    Height = 480, ;
                    Left = 408, ;
                    Top = 24, ;
                    Width = 618, ;
                    Name = "Image2"
ADD OBJECT piesa_generalizata.form3.command1 AS
commandbutton WITH ;
                    Top = 552, ;
                    Left = 24, ;
                    Height = 36, ;
                    Width = 168, ;
                    Caption = "Freza tangentiala ansamblu", ;
                    Name = "Command1"
ADD OBJECT piesa_generalizata.form3.command2 AS
commandbutton WITH ;
                    Top = 552, ;
                    Left = 936, ;
                    Height = 37, ;
                    Width = 49, ;
                    Caption = "Back", ;
                    Name = "Command2"
ADD OBJECT piesa_generalizata.form3.image1 AS image
WITH ;
                    Picture = "freza_tangentiala_1.jpg", ;
                    Stretch = 1, ;
                    Height = 492, ;
                    Left = 24, ;
                    Top = -24, ;
                    Width = 408, ;
                    Name = "Image1"
ADD OBJECT piesa_generalizata.form3.command3 AS
commandbutton WITH ;
                    Top = 552, ;
                    Left = 444, ;
                    Height = 36, ;
                    Width = 204, ;
                    Caption = "Suprafete operatii", ;
                    Name = "Command3"
ADD OBJECT piesa_generalizata.form3.image3 AS image
WITH ;
                    Picture = "supraf_tang.jpg", ;
                    Height = 369, ;
                    Left = 60, ;
                    Top = 120, ;
                    Visible = .F., ;
                    Width = 497, ;
                    Name = "Image3"
ADD OBJECT piesa_generalizata.form3.image4 AS image
WITH ,
                    Picture = "operatii_tangentiale.jpg", ;
                    Height = 436, ;
                    Left = 552, ;
                    Top = 60, ;
                    Visible = .F., ;
                    Width = 467, ;
                    Name = "Image4"
          ADD OBJECT form4 AS form WITH ;
                    Top = 0, ;
                    Left = 0, ;
                    Height = 742, ;
                    Width = 1017, ;
                    DoCreate = .T., ;
Caption = "fREZA TANGENTIALA ANSAMBLU", ;
                    WindowState = 2, ;
                    BackColor = RGB(0,0,0), ;
                    Name = "Form4"
ADD OBJECT piesa_generalizata.form4.image1 AS image
WITH ;
                    Picture =
"freza_tangentiala_ansamblu.jpg", ;
                    Height = 619, ;
                    Left = 24, ;
                    Top = 0, ;
                    Width = 984, ;
                    Name = "Image1"
ADD OBJECT piesa_generalizata.form4.command1 AS
commandbutton WITH ;
                    Top = 480, ;
                    Left = 60, ;
                    Height = 37, ;
                    Width = 61, ;
                    Caption = "Back", ;
                    Name = "Command1"
ADD OBJECT form5 AS form WITH ;
                    Top = 0, ;
                    Left = 0, ;
                    Height = 742, ;
                    Width = 1017, ;
                    DoCreate = .T., ;
                    Caption = "Freza radiala", ;
                    WindowState = 2, ;
                    BackColor = RGB(0,0,0), ;
                    Name = "Form5"
ADD OBJECT piesa_generalizata.form5.image2 AS image
WITH ;
                    Picture = "freza_radiala_executie_1.jpg", ;
                    Stretch = 1, ;
                    Height = 504, ;
                    Left = 384, ;
                    Top = 0, ;
                    Width = 636, ;
                    Name = "Image2"
          ADD OBJECT piesa_generalizata.form5.image1 AS
image WITH ;
                    Picture = "freza_radiala_2.jpg", ;
                    Stretch = 1, ;
                    Height = 432, ;
                    Left = 12, ;
                    Top = 0, ;
                    Width = 456, ;
                    Name = "Image1"
ADD OBJECT piesa_generalizata.form5.command3 AS
commandbutton WITH ;
                    Top = 540, ;
                    Left = 24, ;
                    Height = 36, ;
                    Width = 168, ;
                    Caption = "Freza radiala ansamblu", ;
                    Name = "Command3"
ADD OBJECT piesa_generalizata.form5.command1 AS
commandbutton WITH ;
                    Top = 540, ;
                    Left = 912, ;
                    Height = 37, ;
                    Width = 49, ;
                    Caption = "Back", ;
                    Name = "Command1"
ADD OBJECT piesa_generalizata.form5.command2 AS
commandbutton WITH ;
                    Top = 540, ;
                    Left = 396, ;
                    Height = 36, ;
                    Width = 204, ;
                    Caption = "Suprafete operatii", ;
                    Name = "Command2"
ADD OBJECT piesa_generalizata.form5.image3 AS image
WITH ;
```

```
                Picture = "supraf_radial.jpg", ;
                Height = 385, ;
                Left = 48, ;
                Top = 36, ;
                Visible = .F., ;
                Width = 508, ;
                Name = "Image3"
ADD OBJECT piesa_generalizata.form5.image4 AS image
WITH ;
                Picture = "operatii_radial.jpg", ;
                Height = 400, ;
                Left = 552, ;
                Top = 24, ;
                Visible = .F., ;
                Width = 454, ;
                Name = "Image4"
        ADD OBJECT form6 AS form WITH ;
                Top = 0, ;
                Left = 0, ;
                Height = 742, ;
                Width = 1017, ;
                DoCreate = .T., ;
        Caption = "FREZA RADIALA ANSAMBLU", ;
                WindowState = 2, ;
                BackColor = RGB(0,0,0), ;
                Name = "Form6"
ADD OBJECT piesa_generalizata.form6.image1 AS image
WITH ;
                Picture = "freza_radiala_ansamblu.jpg", ;
                Height = 611, ;
                Left = 24, ;
                Top = 12, ;
                Width = 1008, ;
                BorderColor = RGB(0,0,0), ;
                Name = "Image1"
ADD OBJECT piesa_generalizata.form6.command2 AS
commandbutton WITH ;
                Top = 480, ;
                Left = 60, ;
                Height = 37, ;
                Width = 61, ;
                Caption = "Back", ;
                Name = "Command2"
ADD OBJECT form7 AS form WITH ;
                Top = 33, ;
                Left = 238, ;
                Height = 357, ;
                Width = 448, ;
                DoCreate = .T., ;
Caption = "OPERATIA 2:STRUNJIRE DE DEGROSARE
FRONTALA", ;
                Visible = .F., ;
                AlwaysOnTop = .T., ;
                ForeColor = RGB(0,0,0), ;
                BackColor = RGB(0,0,0), ;
                Name = "Form7"
ADD OBJECT piesa_generalizata.form7.label1 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "a) prindere Sf", ;
                Height = 24, ;
                Left = 12, ;
                Top = 24, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label1"
ADD OBJECT piesa_generalizata.form7.label2 AS label WITH;
                FontBold = .T., ;
                BackStyle = 0, ;
        Caption = "1. Strunjire frontala de degrosare", ;
                Height = 25, ;
                Left = 48, ;
                Top = 48, ;
                Visible = .F., ;
                Width = 192, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label2"
ADD OBJECT piesa_generalizata.form7.label3 AS label WITH,
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "b) desprindere Sf", ;
                Height = 24, ;
                Left = 12, ;
                Top = 84, ;
                Visible = .F., ;
                Width = 144, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label3"
ADD OBJECT piesa_generalizata.form7.label4 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "c) control Sf", ;
                Height = 24, ;
                Left = 12, ;
                Top = 120, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label4"
ADD OBJECT piesa_generalizata.form7.image1 AS image
WITH ;
                Picture = "operatia2.jpg", ;
                Height = 166, ;
                Left = 180, ;
                Top = 84, ;
                Visible = .F., ;
                Width = 174, ;
                Name = "Image1"
ADD OBJECT form8 AS form WITH ;
                Top = 35, ;
                Left = 246, ;
                Height = 352, ;
                Width = 437, ;
                DoCreate = .T., ;
                Caption = "OPERATIA3: STRUNJIRE
DE DEGROSARE CILINDRICA", ;
                Visible = .F., ;
                AlwaysOnTop = .T., ;
                BackColor = RGB(0,0,0), ;
                Name = "Form8"
ADD OBJECT piesa_generalizata.form8.label1 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "a) prindere Sf", ;
                Height = 24, ;
                Left = 20, ;
                Top = 32, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label1"
ADD OBJECT piesa_generalizata.form8.label2 AS label WITH;
                FontBold = .T., ;
                BackStyle = 0, ;
        Caption = "1. Strunjire cilindrica de degrosare", ;
```

```
                Height = 25, ;
                Left = 56, ;
                Top = 56, ;
                Visible = .F., ;
                Width = 220, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label2"
ADD OBJECT piesa_generalizata.form8.label3 AS label WITH
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "b) desprindere Sf", ;
                Height = 24, ;
                Left = 20, ;
                Top = 92, ;
                Visible = .F., ;
                Width = 144, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label3"
ADD OBJECT piesa_generalizata.form8.label4 AS label WITH
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "c) control Sf", ;
                Height = 24, ;
                Left = 20, ;
                Top = 128, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label4"
ADD OBJECT piesa_generalizata.form8.image1 AS image
WITH ;
                Picture = "operatia3.jpg", ;
                Height = 166, ;
                Left = 188, ;
                Top = 92, ;
                Visible = .F., ;
                Width = 174, ;
                Name = "Image1"
        ADD OBJECT form9 AS form WITH ;
                Top = 39, ;
                Left = 250, ;
                Height = 346, ;
                Width = 429, ;
                DoCreate = .T., ;
Caption = "OPERATIA4: STRUNJIRE DE DEGROSARE
FRONTALA", ;
                Visible = .F., ;
                AlwaysOnTop = .T., ;
                BackColor = RGB(0,0,0), ;
                Name = "Form9"
ADD OBJECT piesa_generalizata.form9.label1 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "a) prindere Sf", ;
                Height = 24, ;
                Left = 28, ;
                Top = 40, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label1"
ADD OBJECT piesa_generalizata.form9.label2 AS label WITH;
                FontBold = .T., ;
                BackStyle = 0, ;
        Caption = "1. Strunjire frontala de degrosare", ;
                Height = 25, ;
                Left = 64, ;
                Top = 64, ;
                Visible = .F., ;
                Width = 220, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label2"


ADD OBJECT piesa_generalizata.form9.label3 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "b) desprindere Sf", ;
                Height = 24, ;
                Left = 28, ;
                Top = 100, ;
                Visible = .F., ;
                Width = 144, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label3"
ADD OBJECT piesa_generalizata.form9.label4 AS label WITH;
                FontBold = .T., ;
                FontSize = 12, ;
                WordWrap = .T., ;
                BackStyle = 0, ;
                Caption = "c) control Sf", ;
                Height = 24, ;
                Left = 28, ;
                Top = 136, ;
                Visible = .F., ;
                Width = 108, ;
                ForeColor = RGB(255,255,255), ;
                Name = "Label4"
ADD OBJECT piesa_generalizata.form9.image1 AS image
WITH ;
                Picture = "operatia4.jpg", ;
                Height = 172, ;
                Left = 196, ;
                Top = 100, ;
                Visible = .F., ;
                Width = 154, ;
                Name = "Image1"
        PROCEDURE command1.Click
                thisformset.form1.Visible=.f.
                thisformset.form3.visible= .F.
                thisformset.form5.Visible=.f.
                thisformset.form6.visible= .F.
                thisformset.form4.visible= .F.
                thisformset.form2.Visible=.t.
                thisformset.form2.command1.Visible=.t.
        ENDPROC
        PROCEDURE commandgroup1.Click
                thisformset.form1.Visible=.f.
                thisformset.form2.visible= .F.
                thisformset.form4.visible= .F.
                thisformset.form3.Visible=.t.
                thisformset.form3.command1.Visible=.t.
        ENDPROC
        PROCEDURE commandgroup2.Click
                thisformset.form1.Visible=.f.
                thisformset.form2.visible= .F.
                thisformset.form4.visible= .F.
                thisformset.form3.Visible=.t.
                thisformset.form3.command1.Visible=.t.
        ENDPROC
        PROCEDURE commandgroup3.Click
                thisformset.form1.Visible=.f.
                thisformset.form2.visible= .F.
                thisformset.form4.visible= .F.
                thisformset.form3.Visible=.t.
                thisformset.form3.command1.Visible=.t.
        ENDPROC
        PROCEDURE commandgroup4.Click
```

```
        thisformset.form1.Visible=.f.
        thisformset.form2.visible= .F.
        thisformset.form4.visible= .F.
        thisformset.form3.Visible=.t.
        thisformset.form3.command1.Visible=.t.
ENDPROC
PROCEDURE commandgroup5.Click
        thisformset.form1.Visible=.f.
        thisformset.form2.visible= .F.
        thisformset.form4.visible= .F.
        thisformset.form5.Visible=.t.
        thisformset.form6.visible= .F.
        thisformset.form3.visible= .F.
        thisformset.form5.command1.Visible=.t.
ENDPROC
PROCEDURE command2.RightClick
        thisformset.form1.image4.visible=.f.
        thisformset.form1.image3.visible=.f.

thisformset.form1.commandgroup7.visible=.t.
ENDPROC
PROCEDURE command2.Click
        thisformset.form1.image3.visible=.t.
        thisformset.form1.image4.visible=.t.

thisformset.form1.commandgroup6.visible=.t.

thisformset.form1.commandgroup7.visible=.t.

thisformset.form1.commandgroup8.visible=.t.
ENDPROC
PROCEDURE commandgroup6.Click
        *thisformset.form1.visible=.f.
        thisformset.form2.visible=.f.
        thisformset.form3.visible=.f.
        thisformset.form4.visible=.f.
        thisformset.form5.visible=.f.
        thisformset.form6.visible=.f.
        thisformset.form7.visible=.t.
        thisformset.form7.label1.visible=.t.
        thisformset.form7.label2.visible=.t.
        thisformset.form7.label3.visible=.t.
        thisformset.form7.label4.visible=.t.
        thisformset.form7.image1.visible=.t.
ENDPROC
PROCEDURE commandgroup6.RightClick
        thisformset.form7.visible=.f.
        thisformset.form7.label1.visible=.f.
        thisformset.form7.label2.visible=.f.
        thisformset.form7.label3.visible=.f.
        thisformset.form7.label4.visible=.f.
        thisformset.form7.image1.visible=.f.
ENDPROC
PROCEDURE commandgroup7.Click
        thisformset.form2.visible=.f.
        thisformset.form3.visible=.f.
        thisformset.form4.visible=.f.
        thisformset.form5.visible=.f.
        thisformset.form6.visible=.f.
        thisformset.form8.visible=.t.
        thisformset.form8.label1.visible=.t.
        thisformset.form8.label2.visible=.t.
        thisformset.form8.label3.visible=.t.
        thisformset.form8.label4.visible=.t.
        thisformset.form8.image1.visible=.t.
ENDPROC
PROCEDURE commandgroup7.RightClick
        thisformset.form8.visible=.f.
        thisformset.form8.label1.visible=.f.
        thisformset.form8.label2.visible=.f.
        thisformset.form8.label3.visible=.f.
        thisformset.form8.label4.visible=.f.
        thisformset.form8.image1.visible=.f.
ENDPROC
PROCEDURE commandgroup8.RightClick
        thisformset.form9.visible=.f.
        thisformset.form9.label1.visible=.f.
        thisformset.form9.label2.visible=.f.
        thisformset.form9.label3.visible=.f.
        thisformset.form9.label4.visible=.f.
        thisformset.form9.image1.visible=.f
ENDPROC
PROCEDURE commandgroup8.Click
        thisformset.form2.visible=.f.
        thisformset.form3.visible=.f.
        thisformset.form4.visible=.f.
        thisformset.form5.visible=.f.
        thisformset.form6.visible=.f.
        thisformset.form9.visible=.t.
        thisformset.form9.label1.visible=.t.
        thisformset.form9.label2.visible=.t.
        thisformset.form9.label3.visible=.t.
        thisformset.form9.label4.visible=.t.
        thisformset.form9.image1.visible=.t.
ENDPROC
PROCEDURE commandgroup9.Click
        thisformset.form1.Visible=.f.
        thisformset.form2.visible= .F.
        thisformset.form4.visible= .F.
        thisformset.form5.Visible=.t.
        thisformset.form6.visible= .F.
        thisformset.form3.visible= .F.
        thisformset.form5.command1.Visible=.t.
ENDPROC
PROCEDURE commandgroup10.Click
        thisformset.form1.Visible=.f.
        thisformset.form2.visible= .F.
        thisformset.form4.visible= .F.
        thisformset.form5.Visible=.t.
        thisformset.form6.visible= .F.
        thisformset.form3.visible= .F.
        thisformset.form5.command1.Visible=.t.
ENDPROC
PROCEDURE command3.Click
        thisformset.release
        clear events
ENDPROC
PROCEDURE command1.Click
        thisformset.form2.visible= .F.
        thisformset.form3.visible= .F.
        thisformset.form4.visible= .F.
        thisformset.form1.visible= .T.
        thisformset.form2.command1.Visible=.f.
ENDPROC
PROCEDURE command1.Click
        thisformset.form4.visible= .T.
        thisformset.form3.visible= .f.
        thisformset.form4.command1.Visible=.t.
ENDPROC
PROCEDURE command2.Click
        thisformset.form3.visible= .F
        thisformset.form1.visible= .T.
        thisformset.form3.command1.Visible=.f.
ENDPROC
PROCEDURE command3.Click
        thisformset.form3.image3.visible=.t.
        thisformset.form3.image4.visible=.t.

*thisformset.form1.commandgroup6.visible=.t.

*thisformset.form1.commandgroup7.visible=.t.

*thisformset.form1.commandgroup8.visible=.t.
ENDPROC
```

```
        PROCEDURE command3.RightClick
                thisformset.form3.image4.visible=.f.
                thisformset.form3.image3.visible=.f.

        *thisformset.form1.commandgroup7.visible=.t.
        ENDPROC
        PROCEDURE command1.Click
                thisformset.form4.visible= .F.
                thisformset.form1.visible= .F.
                thisformset.form2.visible= .F.
                thisformset.form3.visible= .T.
                thisformset.form4.command1.Visible=.f.
        ENDPROC
        PROCEDURE command3.Click
                thisformset.form6.visible= .T.
                thisformset.form5.visible= .f.
                thisformset.form4.visible= .f.
                thisformset.form3.visible= .f.
                thisformset.form2.visible= .f.
                thisformset.form1.visible= .f.
                thisformset.form6.command2.Visible=.t.
        ENDPROC
        PROCEDURE command1.Click
.visible=.t.
ENDPROC
        PROCEDURE command2.Click
                thisformset.form4.visible= .F.
                thisformset.form6.visible= .F.
                thisformset.form3.visible= .F.
                thisformset.form3.visible= .F.
                thisformset.form2.visible= .F.
                thisformset.form6.visible= .F.
                thisformset.form4.visible= .F.
                thisformset.form5.visible= .F.
                thisformset.form1.visible= .T.
                thisformset.form5.command1.Visible=.f.
        ENDPROC
        PROCEDURE command2.RightClick
                thisformset.form5.image4.visible=.f.
                thisformset.form5.image3.visible=.f.

        *thisformset.form1.commandgroup7.visible=.t.
        ENDPROC
        PROCEDURE command2.Click
                thisformset.form5.image3.visible=.t.
                thisformset.form5.image4.visible=.t.

        *thisformset.form1.commandgroup6.visible=.t.

        *thisformset.form1.commandgroup7.visible=.t.
                *thisformset.form1.commandgroup8

                thisformset.form1.visible= .F.
                thisformset.form2.visible= .F.
                thisformset.form5.visible= .T.
                thisformset.form6.command2.Visible=.f.
        ENDPROC
ENDDEFINE
```

8.4.2.5. Schemele externe a bazei de date obiectuale "Sabloane"

Schema externă pentru clasele care generează baza de date obiectuală "Şabloane" şi cele care gestionează informaţia din baza de date "Şabloane" sunt prezentate în figura 8.44.

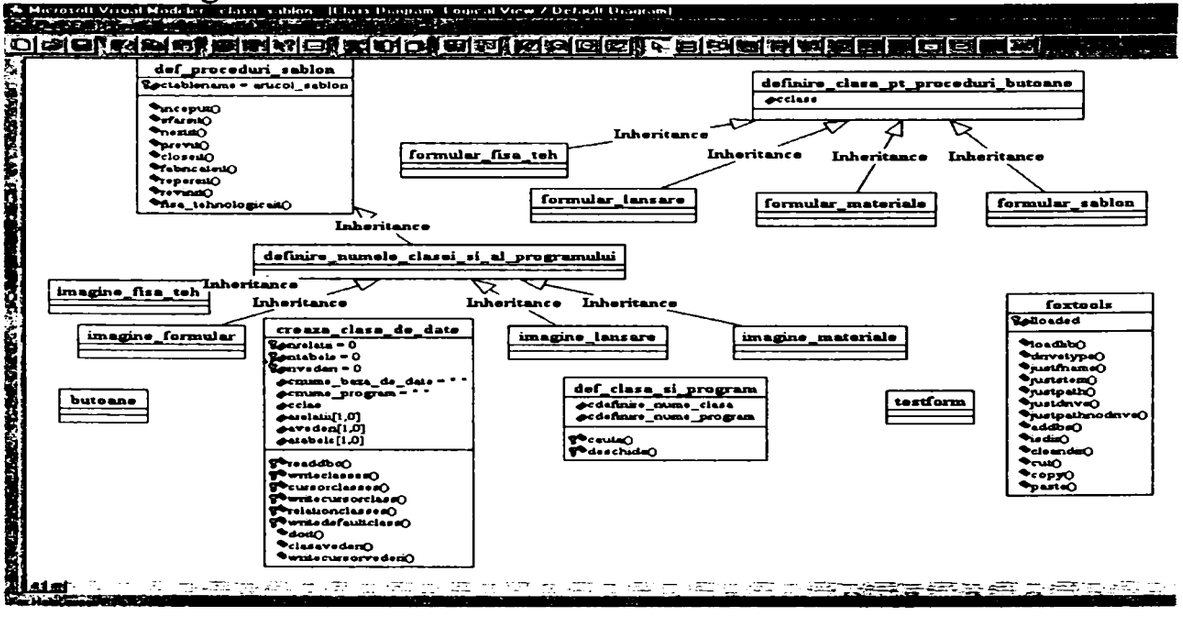


Fig 8.44 Schema externă pentru clasele care generează baza de date obiectuală "Şabloane"

Schema externă pentru baza de date "Şabloane" este prezentată în figura 8.45.

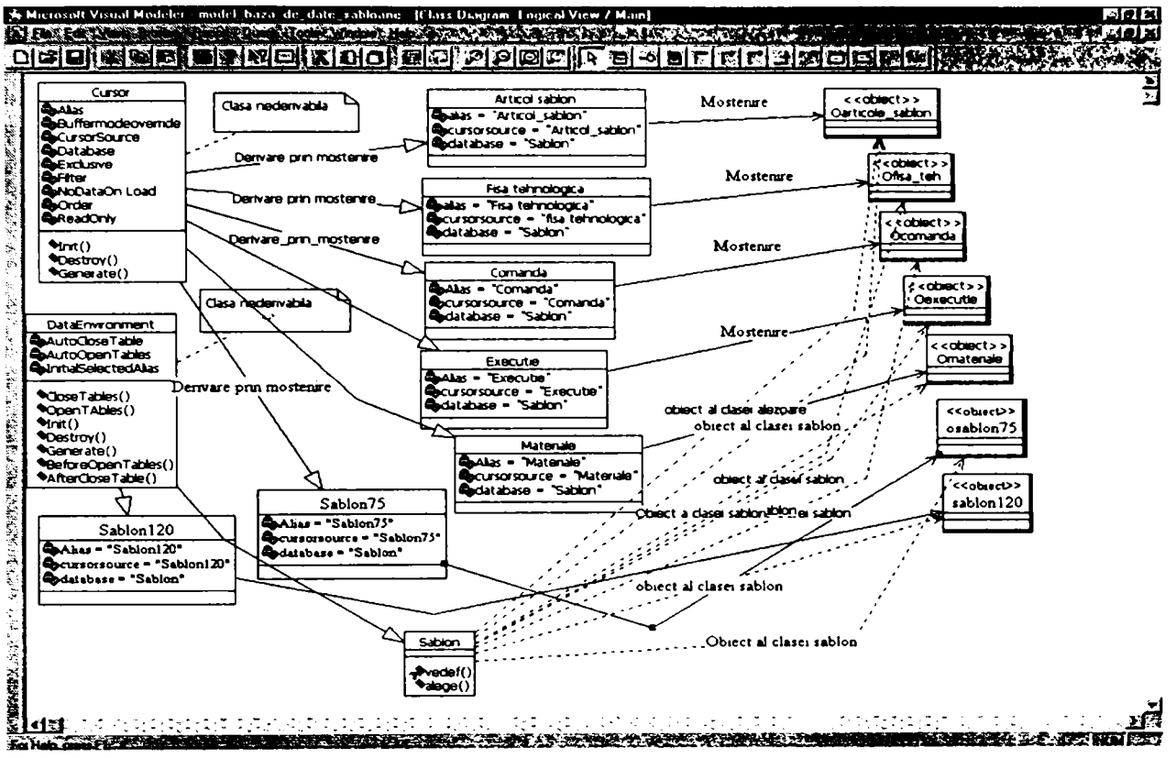


Fig 8.45 Schema externă pentru baza de date "Şabloane"

8.4.2.6 Schemele externe a bazei de date obiectuale "STAS"

Schema externă pentru clasele care generează baza de date obiectuală "STAS" şi cele care gestionează informația din baza de date "STAS" sunt prezentate în figura 8.46.

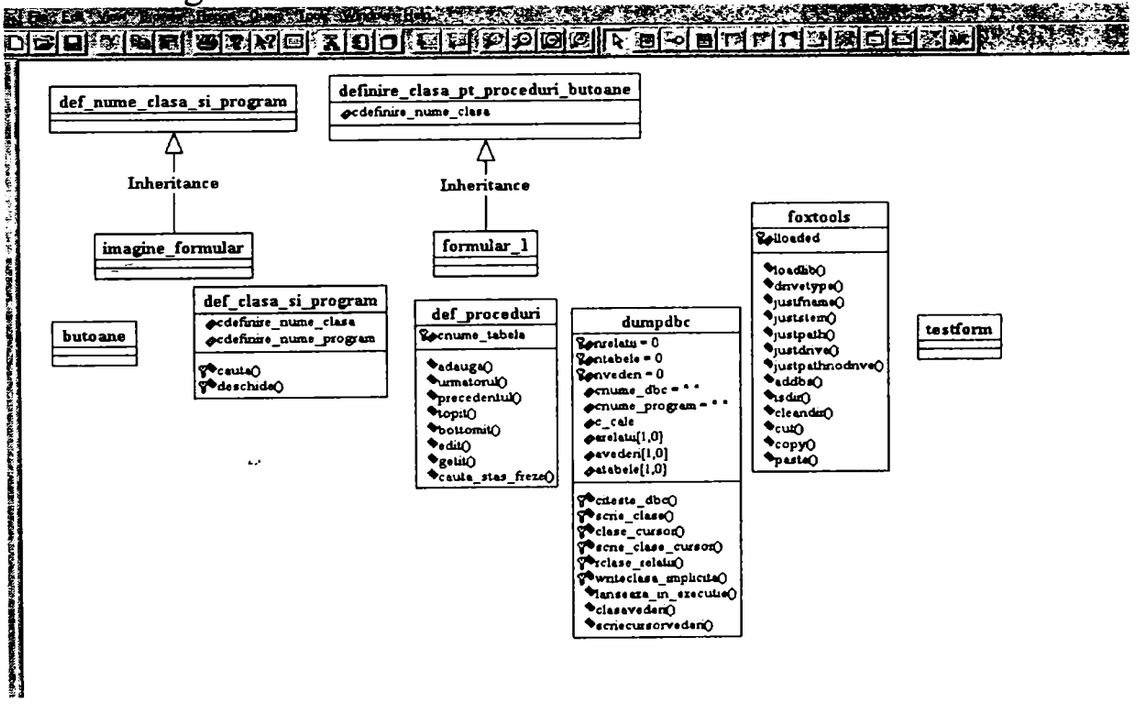


Fig 8.46 Schema externă pentru clasele care generează baza de date obiectuală "STAS"

Schema externă pentru baza de date obiectuală a fost proiectată unitar pentru tipurile de scule importante:freze, tarozi, burghie, cuțite, alezoare etc(figurile 8.47-55).

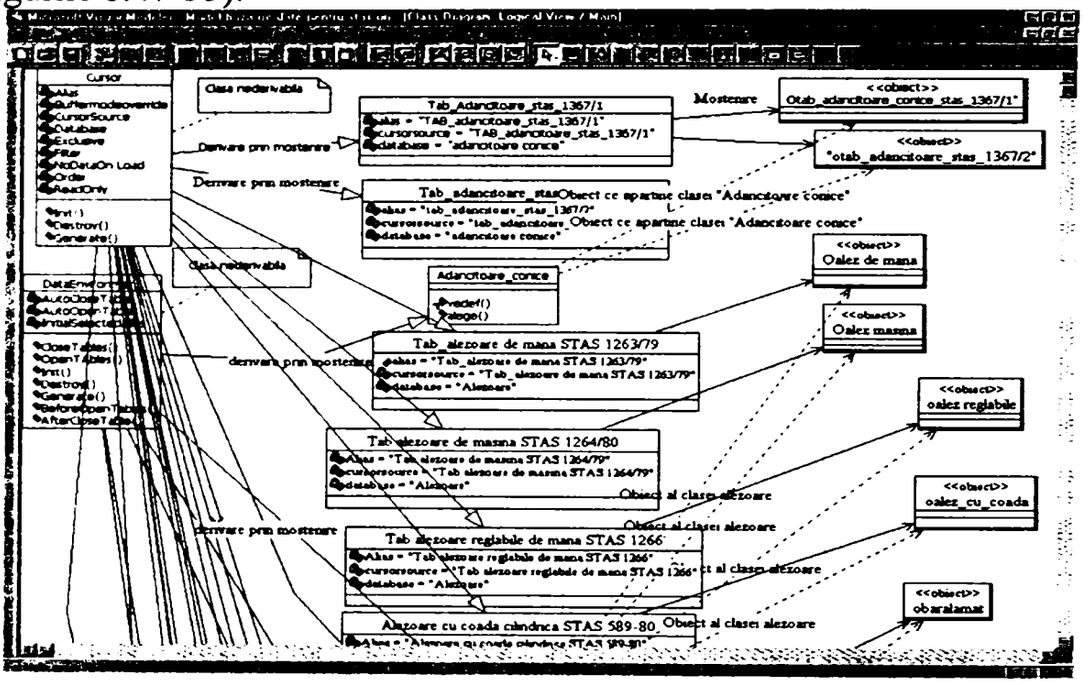


Fig 8.47 Schema externă pentru baza de date "Alezoare"

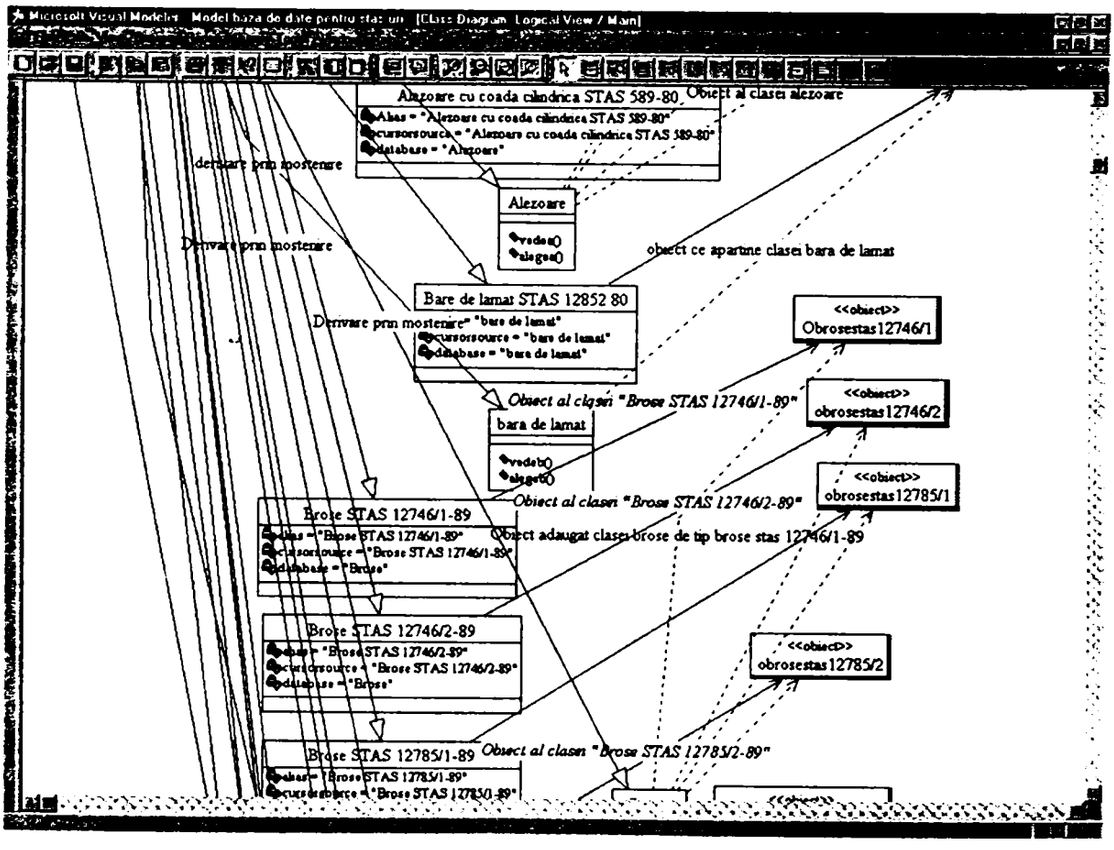


Fig 8.48 Schema externă pentru baza de date "Broşe"

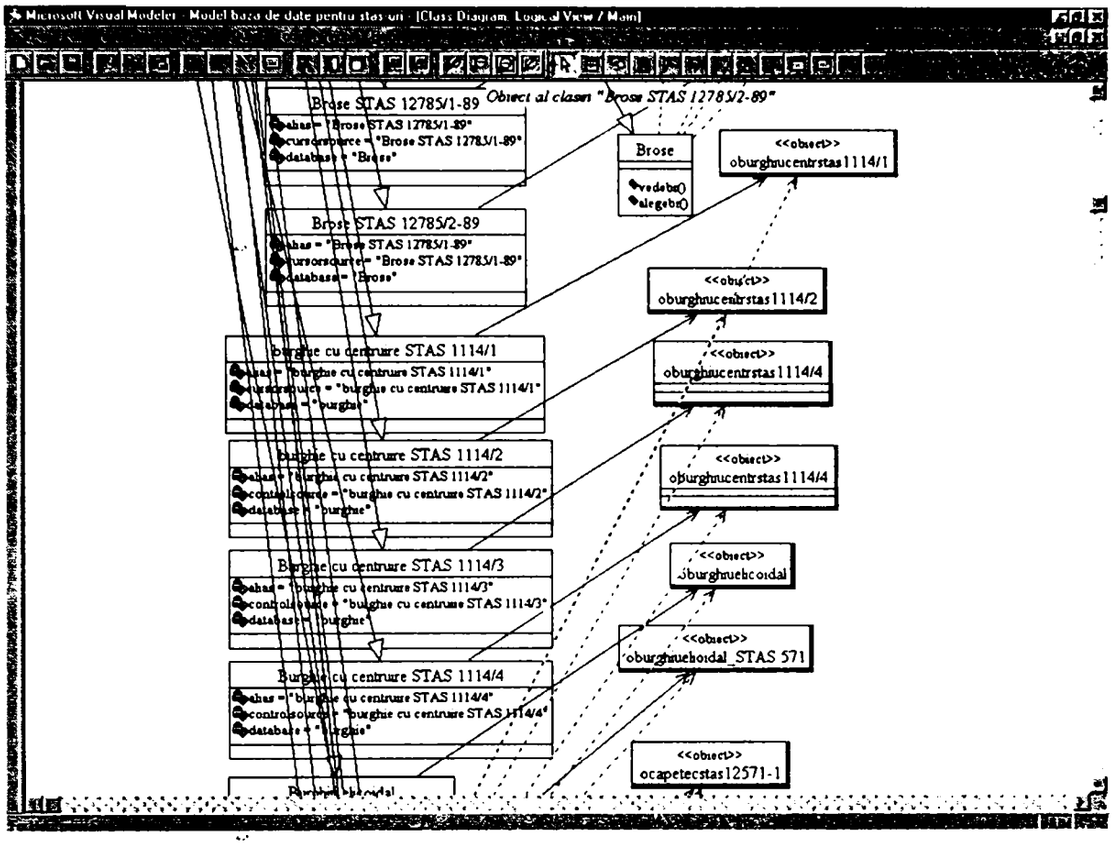


Fig 8.49 Schema externă pentru baza de date "Burghie"

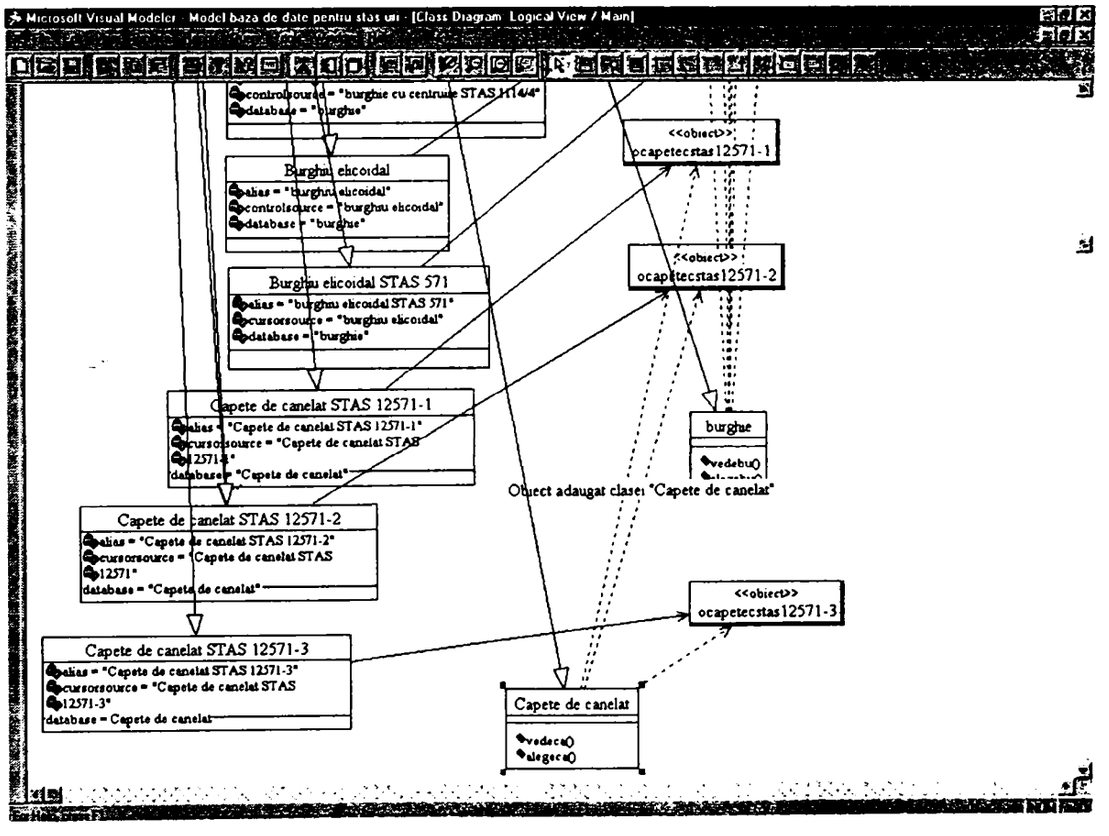


Fig 8.50 Schema externă pentru baza de date "Burghie"

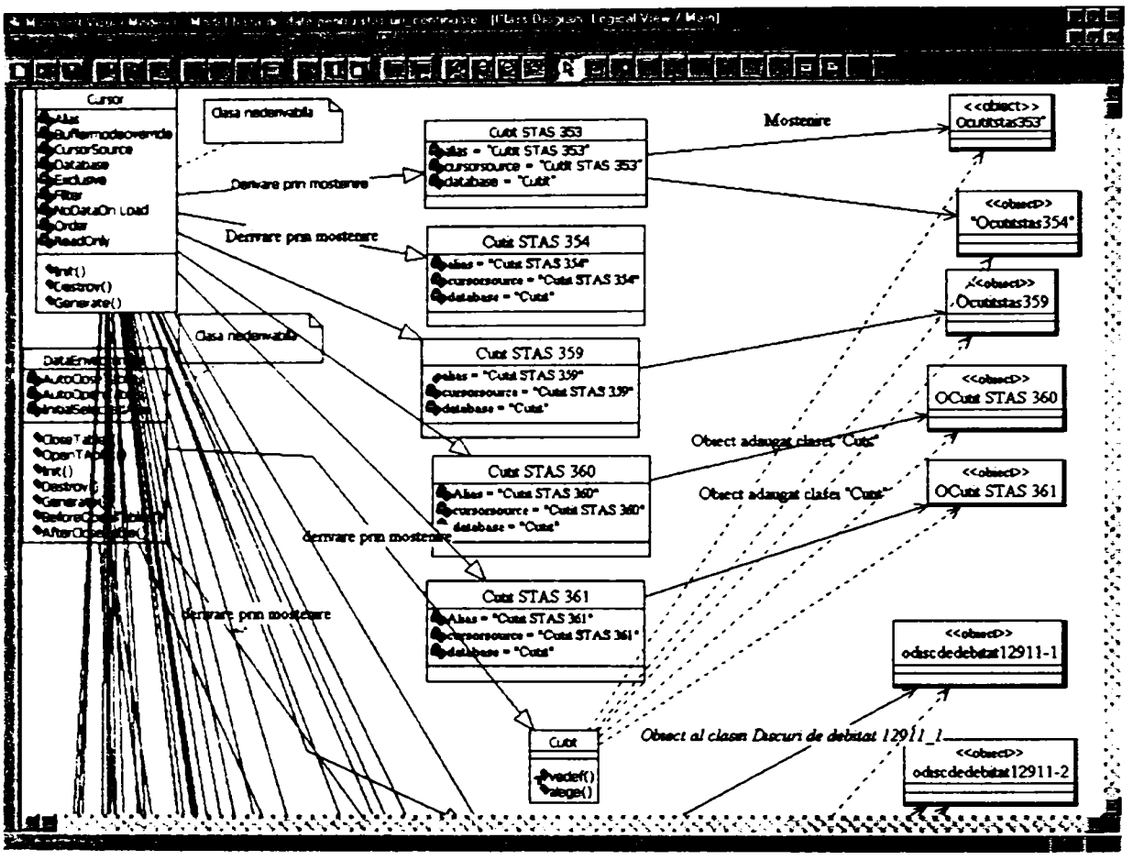


Fig 8.51 Schema externă pentru baza de date "Cutite"

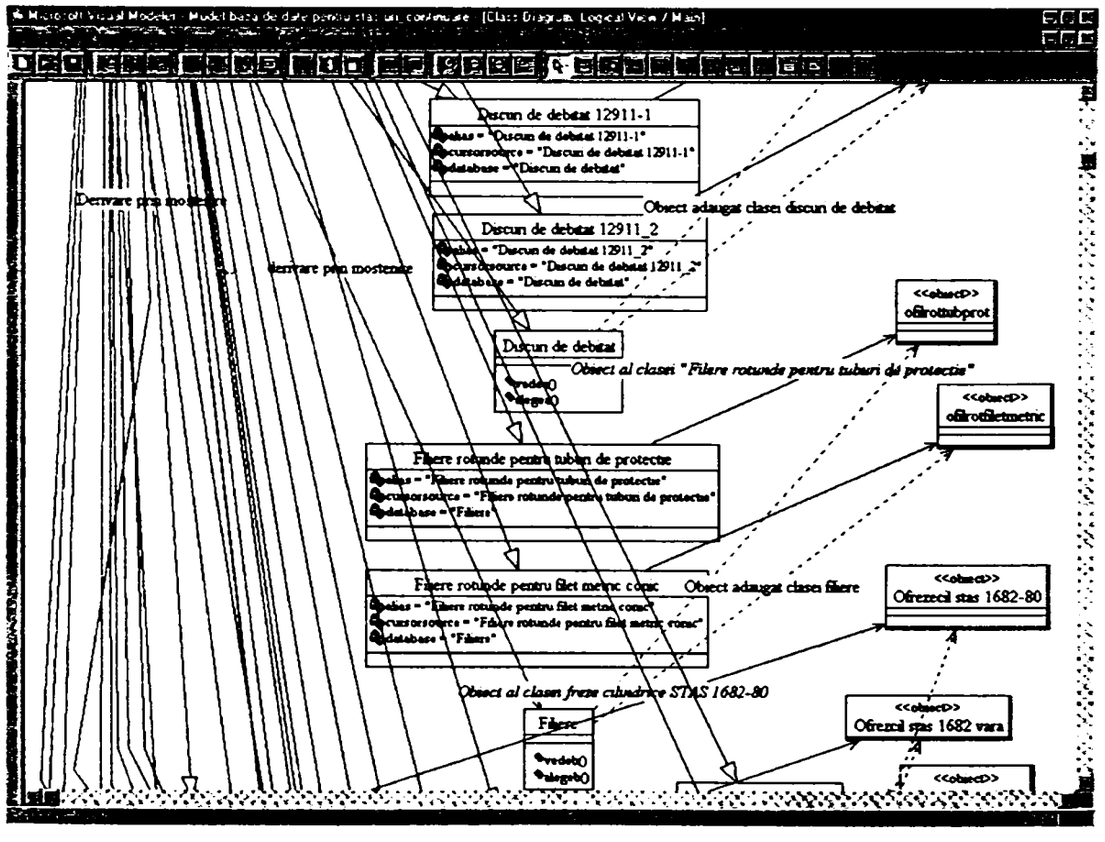


Fig.8.52 Schema externă pentru baza de date "Freze"

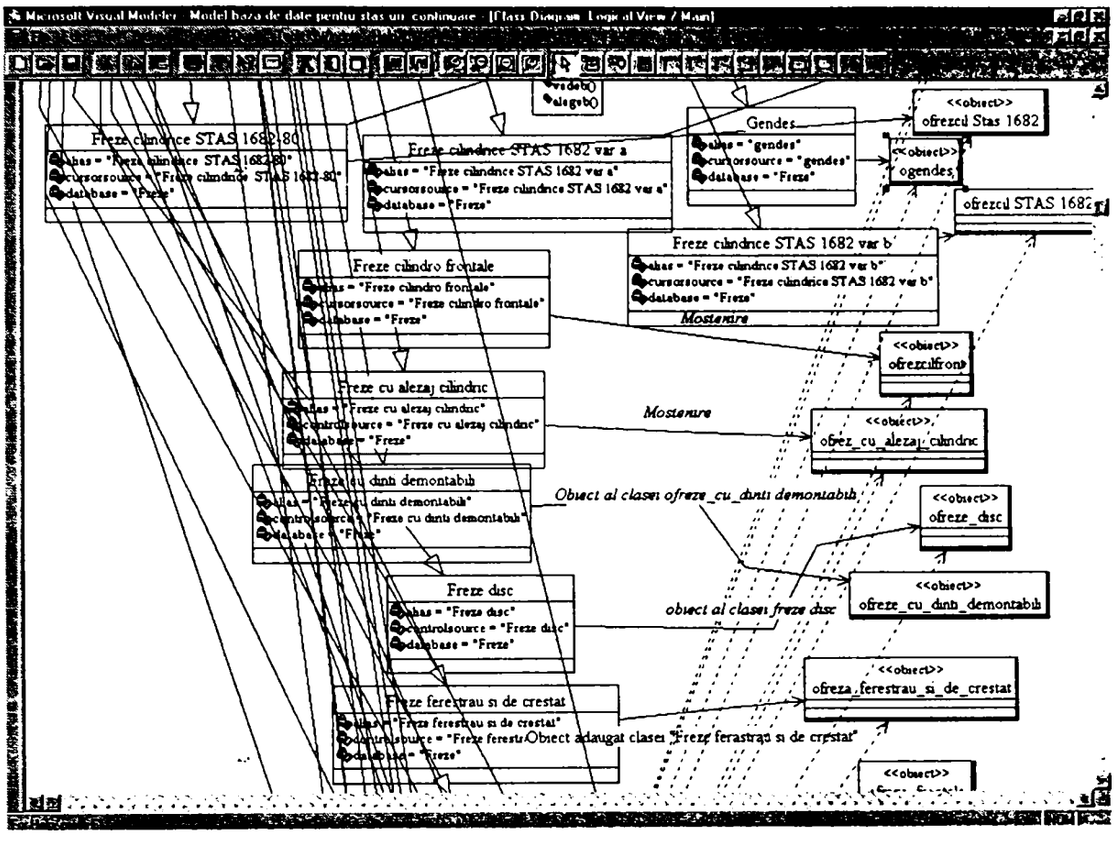


Fig 8.53 Schema externă pentru baza de date "Freze"

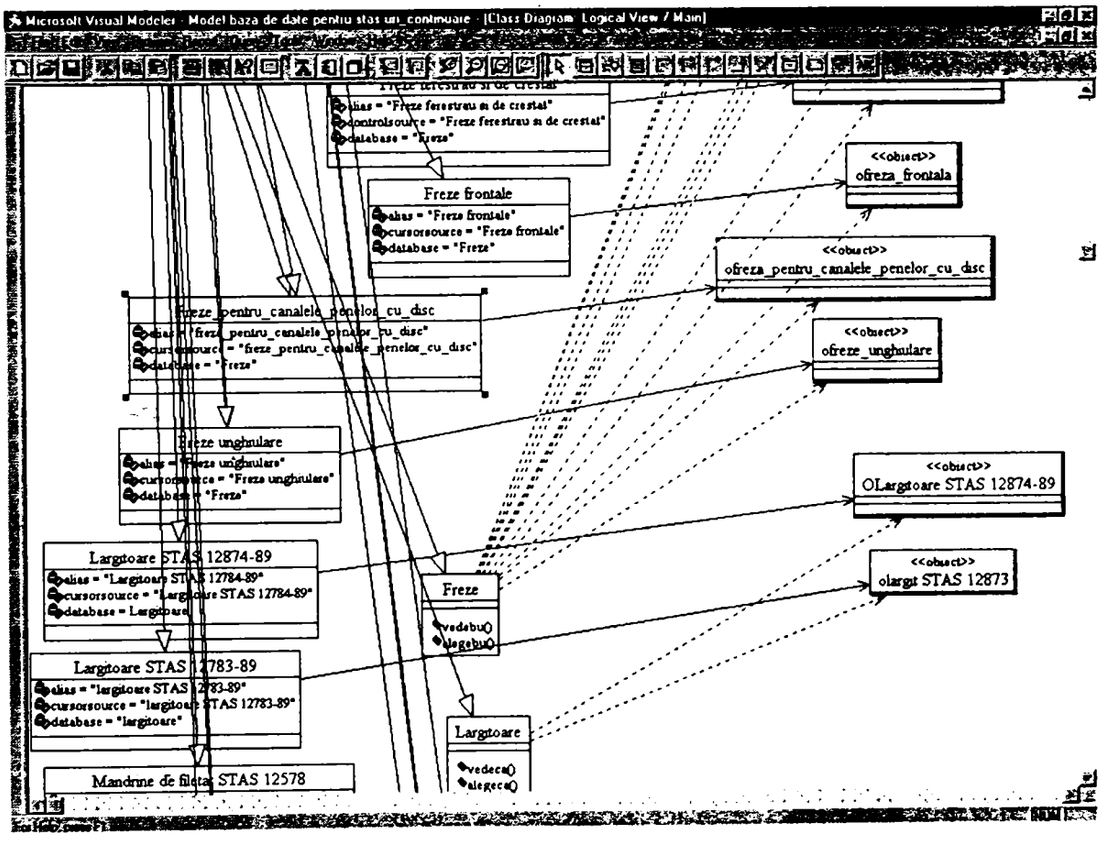


Fig 8.54 Schema externă pentru baza de date "Freze"

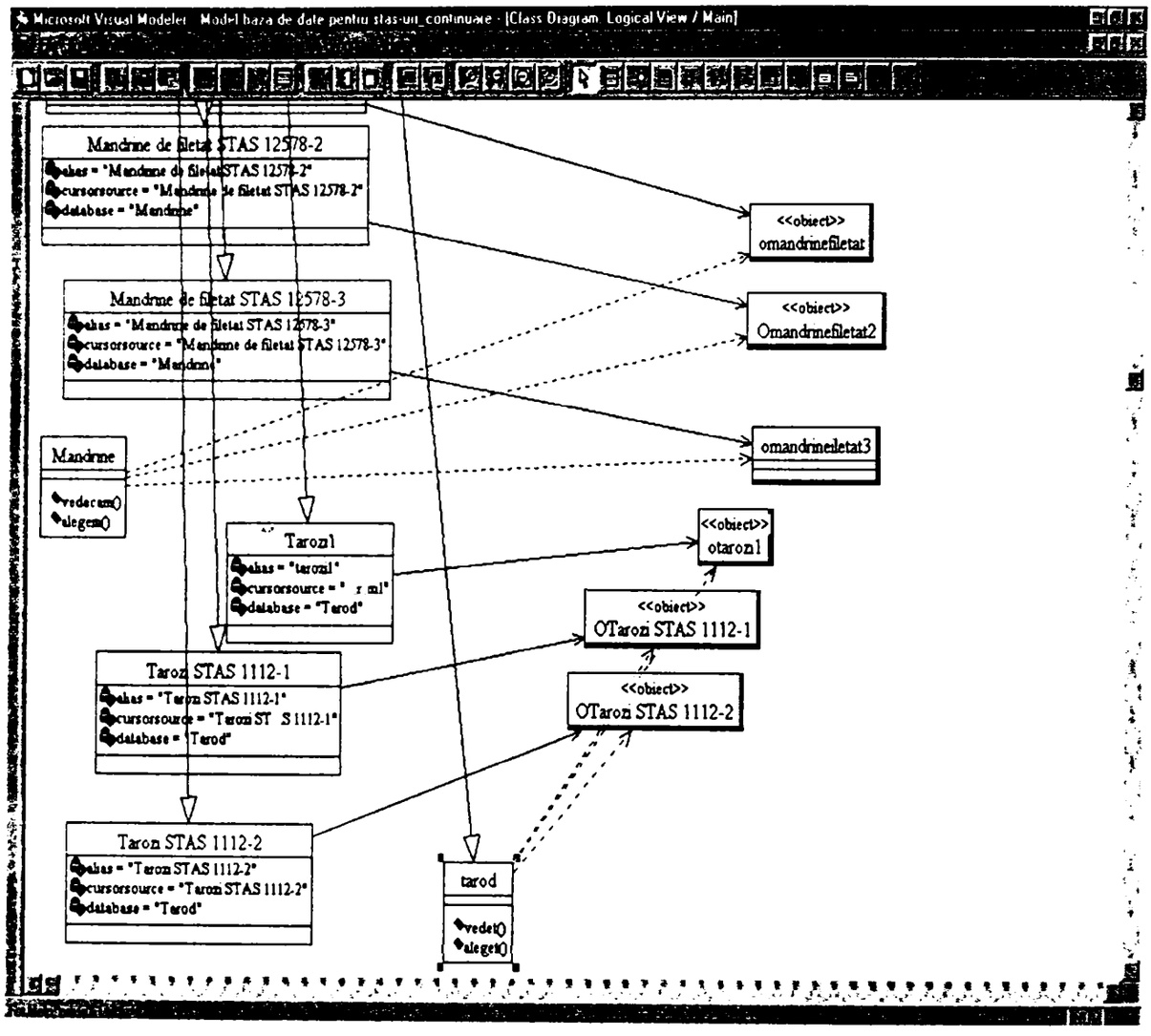


Fig 8.55 Schema externă pentru baza de date "Tarozi"

8.5 Implementarea aplicației pentru Sistemul de Fabricație al Sculelor (SFS)

Întreaga aplicație este implementată obiectual. Dacă din punct de vedere al bazei de date acest lucru a fost arătat anterior, în continuare vor fi prezentate câteva din clasele reprezentative pentru desfăşurarea aplicației. Modul de realizare, păstrează aceleaşi principii. Clasa meniu_cutit este clasă derivabilă. Toate celelalte clase ale aplicației sunt clase derivate prin moştenire. Obiectele claselor meniu participă la apelarea ramurilor aplicației. Meniul principal, este o clasă cu rol de container, din care se vor apela ulterior, celelalte clase derivabile şi derivate (figura 8.56).

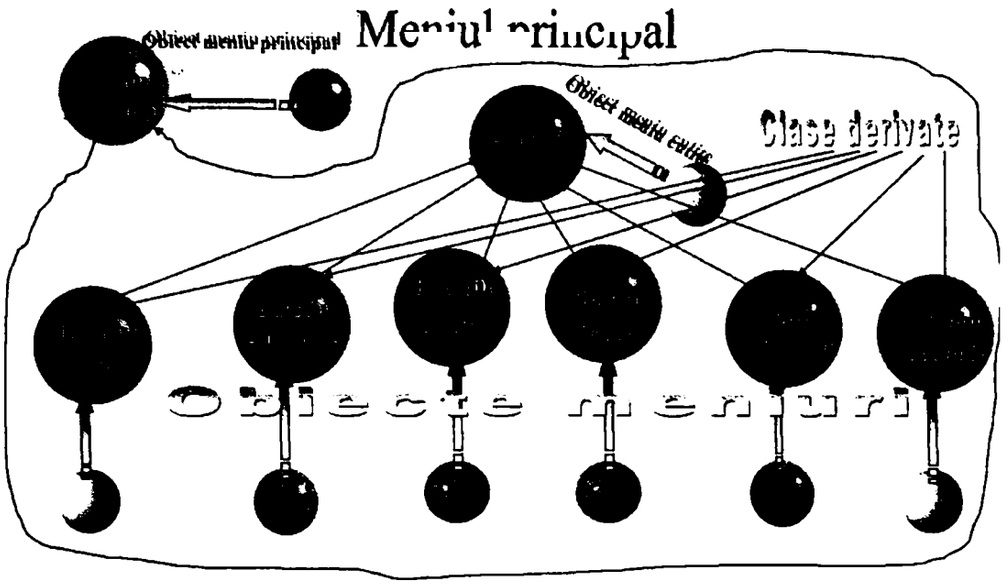


Fig 8.56 Clasele meniuri

În figura 8.57 este prezentată imaginea meniului principal al aplicației.

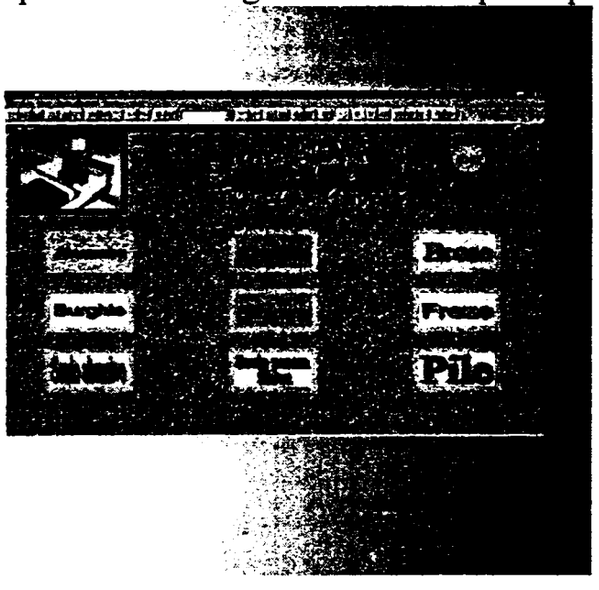


Fig 8.57 Clasa "Meniu Principal"

Programul sursă pentru lansarea aplicației este următorul:

```
set talk off
close all
clea all
set classlib to clase,flatbtn,_multimedia,_base
men=createobject("meniup")
mencutit=createobject("meniu_cutite")
menburghiu=createobject("meniu_burghie")
menfreza=createobject("meniu_freze")
menbrosa=createobject("meniu_brose")
menpila=createobject("meniu_pile")
mensculaf=createobject("meniu_sculaf")
mensculaa=createobject("meniu_sculaa")
mensculad=createobject("meniu_sculad")
set default to d:\sorin\doctorat\teza_de_doctorat\alezor\rez
set classlib to clase_aplicatie,flatbtn,plan_de_operatii
                oform=createobject("formular")
                opl=createobject("plan1")
set default to d:\sorin\doctorat\teza_de_doctorat\meniu
set classlib to clase,flatbtn,_multimedia,_base
men.show
read events

define class meniu_burghie as meniu_cutite
        cmdshadow1.Image.Picture = "brg-el1.gif"
        cmdshadow2.Image.Picture = "brg-el1.gif"
        cmdshadow3.Image.Picture = "brg-el1.gif"
        video1.videofile="text_burghie.avi"
        caption='MENIU SCULE DE GAURIT'
procedure cmdshadow3.click
set default to d:\sorin\doctorat\teza_de_doctorat\alezor\rez
```

```
set classlib to clase_aplicatie.flatbtn,plan_de_operatii
                    oform.show()
                    read events
set default to d:\sorin\doctorat\teza_de_doctorat\meniu
endproc
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh
define class meniu_freze as meniu_cutite
                    cmdshadow1.Image.Picture = "freza.gif"
                    cmdshadow2.Image.Picture = "freza.gif"
                    cmdshadow3.Image.Picture = "freza.gif"
                    video1.videofile="text_freze.avi"
                    caption='MENIU FREZE'
                    picture='snowgs.jpg'
procedure cmdshadow1.click
set default to d:\sorin\doctorat\teza_de_doctorat\stas
                    set classlib to clase
                    oform=createobject("formular_1")
                    oform.show()
                    read events
set default to d:\sorin\doctorat\teza_de_doctorat\meniu
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh
define class meniu_brose as meniu_cutite
                    cmdshadow1.Image.Picture = "brosa.gif"
                    cmdshadow2.Image.Picture = "brosa.gif"
                    cmdshadow3.Image.Picture = "brosa.gif"
                    video1.videofile="text_brosa.avi"
                    caption='MENIU BROSE'
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh
define class meniu_pile as meniu_cutite
                    cmdshadow1.Image.Picture = ""
                    cmdshadow2.Image.Picture = ""
                    cmdshadow3.Image.Picture = ""
                    video1.videofile="pile.avi"
                    caption='MENIU PILE'
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh

define class meniu_sculaf as meniu_cutite
                    cmdshadow1.Image.Picture = ""
                    cmdshadow2.Image.Picture = ""
                    cmdshadow3.Image.Picture = ""
                    video1.videofile="scule_de_filetat.avi"
          caption='MENIU SCULE PENTRU FILETAT'
                    picture='snowgs.jpg'
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh

define class meniu_sculaa as meniu_cutite
                    cmdshadow1.Image.Picture = ""
                    cmdshadow2.Image.Picture = ""
                    cmdshadow3.Image.Picture = ""
                    video1.videofile="scule_ABRAZIVE.avi"
                    caption='MENIU SCULE ABRAZIVE'
*                   picture='snowgs.jpg'
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh
define class meniu_sculad as meniu_cutite
                    cmdshadow1.Image.Picture = ""
                    cmdshadow2.Image.Picture = ""
                    cmdshadow3.Image.Picture = ""
                    video1.videofile="scule_de_danturat.avi"
          caption='MENIU SCULE DE DANTURAT'
*                   picture='snowgs.jpg'
enddefine
                    cmdshadow2.refresh
                    cmdshadow3.refresh
```

Programul sursă al meniului principal este următorul:

```
*************************************************
DEFINE CLASS meniup AS form
        DataSession = 1
        Top = 0
        Left = 0
        Height = 742
        Width = 1017
        DoCreate = .T.
        ShowTips = .T.
        Picture = "..\imagini_pt_aplicatie\flowers2.jpg"
        Caption = "MENIU"
        FontBold = .T.
        FontSize = 22
        WindowState = 2
        BackColor = RGB(128,128,128)
        Name = "meniup"


        ADD OBJECT shape1 AS shape WITH ;
                Top = 0, ;
                Left = 12, ;
                Height = 181, ;
                Width = 228, ;
                BackColor = RGB(0,0,0), ;
                Name = "Shape1"


        ADD OBJECT command1 AS command WITH ;
                Top = 204, ;
                Left = 72, ;
                Width = 168, ;
                Height = 96, ;
                Picture = "", ;
                BorderWidth = 3, ;
                SpecialEffect = 0, ;
                MouseIcon = "h_point.cur", ;
                BackColor = RGB(192,192,192), ;
                BorderColor = RGB(0,0,0), ;
                Name = "Command1", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
                Image.Picture = "scule_de_gaurit.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT timer_flat1 AS timer_flat WITH ;
```

```
                Top = 312, ;
                Left = 132, ;
                Height = 36, ;
                Width = 84, ;
                Name = "Timer_flat1"
ADD OBJECT _videoplayer1 AS _videoplayer WITH ;
                Top = 48, ;
                Left = 310, ;
                Width = 396, ;
                Height = 133, ;
                Visible = .T., ;
cfilename="d:\sorin\doctorat\teza_de_doctorat\meniu\scule.avi";
                Name = "_VIDEOPLAYER1", ;
                tmrCheckMode.Name = "tmrCheckMode"
        ADD OBJECT command2 AS command WITH ;
                Top = 348, ;
                Left = 72, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command2", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
        Image.Picture = "scule pentru danturat.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT command3 AS command WITH ;
                Top = 216, ;
                Left = 420, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command3", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
                Image.Picture = "text_cutite1.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.MouseIcon = "..\vfox_clase", ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT command4 AS command WITH ,
                Top = 348, ;
                Left = 420, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command4", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
                Image.Picture = "scule abrazive.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ,
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT command6 AS command WITH ;
                Top = 348, ;
                Left = 768, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command6", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
```

```
                Imageoff Name = "Imageoff", ;
                Image Picture = "text_freze1.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT command5 AS command WITH ;
                Top = 216, ;
                Left = 768, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BackColor = RGB(192,192,192), ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command5", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
                Image.Picture = "text_brose.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
        ADD OBJECT command7 AS command WITH ;
                Top = 468, ;
                Left = 240, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command7", ;
                Shadow.ZOrderSet = 1, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 6, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 7, ;
                Imageoff.Name = "Imageoff", ;
                Image.Picture = "pile.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 0, ;
                Label.Name = "Label"
        ADD OBJECT command8 AS command WITH ;
                Top = 468, ;
                Left = 588, ;
                Width = 168, ;
                Height = 96, ;
                BorderWidth = 1, ;
                MouseIcon = "h_point.cur", ;
                BorderColor = RGB(128,128,128), ;
                Name = "Command8", ;
                Shadow.ZOrderSet = 0, ;
                Shadow.Name = "Shadow", ;
                Line4.DefLeft = "", ;
                Line4.DefHeight = "", ;
                Line4.Name = "Line4", ;
                Line2.DefWidth = "", ;
                Line2.Name = "Line2", ;
                Line1.DefTop = "", ;
                Line1.DefWidth = "", ;
                Line1.Name = "Line1", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Shape.ZOrderSet = 5, ;
                Shape.Name = "Shape", ;
                Imageoff.ZOrderSet = 6, ;
                Imageoff.Name = "Imageoff", ;
        Image.Picture = "scule pentru filetat.jpg", ;
                Image.Stretch = 2, ;
                Image.Height = 72, ;
                Image.Left = 12, ;
                Image.Top = 12, ;
                Image.Width = 144, ;
                Image.ZOrderSet = 8, ;
                Image.Name = "Image", ;
                Label.ZOrderSet = 7, ;
                Label.Name = "Label"
ADD OBJECT _videoplayer2 AS _videoplayer WITH ;
                Top = 12, ;
                Left = 24, ;
                Width = 201, ;
                Height = 156, ;
                Visible = .T., ;
        cfilename                                       =
"D:\SORIN\DOCTORAT\TEZA_DE_DOCTORAT\MENIU\K3
0_FERT.AVI", ;
                Name = "_videoplayer2", ;
                tmrCheckMode.Name = "tmrCheckMode"


        ADD OBJECT cmdshadow1 AS cmdshadow WITH ;
                Top = 24, ;
                Left = 840, ;
                Width = 72, ;
                Height = 72, ;
                Name = "Cmdshadow1", ;
                Shadow.Curvature = 99, ;
                Shadow.Name = "Shadow", ;
                Shape.Curvature = 99, ;
                Shape.Name = "Shape", ;
                Line3.DefHeight = "", ;
                Line3.Name = "Line3", ;
                Imageoff.BorderColor                    =
RGB(192,192,192), ;
                Imageoff.Name = "Imageoff", ;
                Line1.DefTop = "", ;
```

```
        Line1.DefWidth = "", ;
        Line1.Name = "Line1", ;
        Line2.DefWidth = "", ;
        Line2.Name = "Line2", ;
        Line4.DefLeft = "", ;
        Line4.DefHeight = "", ;
        Line4.Name = "Line4", ;
        Label.FontBold = .T., ;
        Label.FontSize = 14, ;
        Label.Caption = "EXIT", ;
        Label.Left = 12, ;
        Label.Top = 24, ;
        Label.Name = "Label", ;
        Image.BackStyle = 0, ;
Image.BorderColor = RGB(192,192,192), ;
        Image.Name = "Image"
PROCEDURE command1.Click
                        menburghiu.show
                        read events
ENDPROC

PROCEDURE command2.Click
        mensculad.show
        read events
ENDPROC
PROCEDURE command3.Click
                        mencutit.show
                        read events
ENDPROC
PROCEDURE command3.RightClick
        thisform.list1.visible=.f.
ENDPROC
PROCEDURE command3.Image.RightClick
        thisform.list1.visible=.f.
ENDPROC
PROCEDURE command4.Click
        mensculaa.show
        read events
ENDPROC
PROCEDURE command6.Click
        menfreza.show
        read events
ENDPROC
PROCEDURE command5.Click
        menbrosa.show
        read events
ENDPROC
PROCEDURE command7.Click
        menpila.show
        read events
ENDPROC
PROCEDURE command8.Click
        mensculaf.show
        read events
ENDPROC
PROCEDURE cmdshadow1.Click
        close all
        clear class all
        set classlib to
        clear events
ENDPROC
ENDDEFINE
*************************************************
```

Toate entitățile aplicației sunt obiecte, apelabile în contextul dorit (figura 8.58).

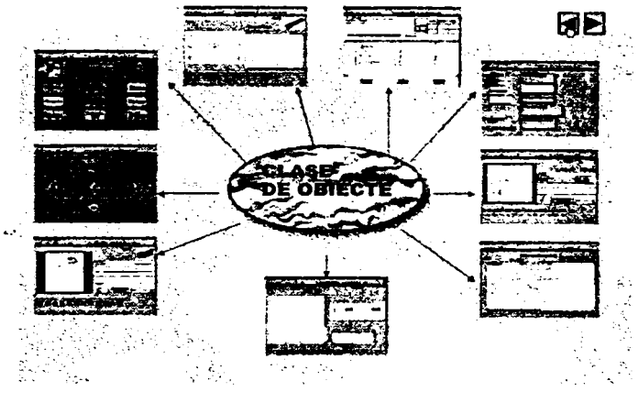


Fig 8.58 Clase de obiecte ale aplicației

Fiecărei ramuri corespunzătoare clasificării generale a sculelor îi corespunde un meniu, din care pot fi apelate trei direcții, corespunzătoare tipului sculei: STAS, fabricată în societatea economică, scule speciale fabricate fie în societate, fie de firme specializate, putând fi consulate în acest scop cataloage de firmă (figura 8.59).

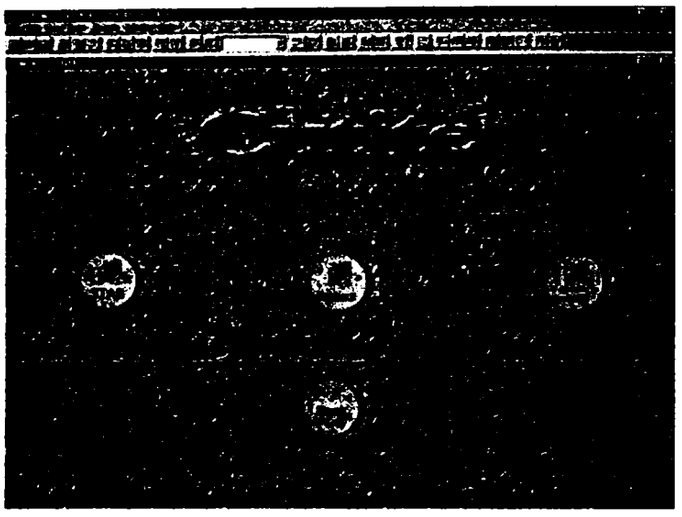


Fig 8.59 Clasa "Meniu Cuțite"

Pentru fiecare categorie de scule există o clasă, care conține obiecte ce permit accesul la datele existente în baza de date orientată obiect. Pe lângă consultarea datelor este posibilă generarea de documente economice (figura 8.60), activități de proiectare, programarea producției, plan de operații, fișa tehnologică etc.

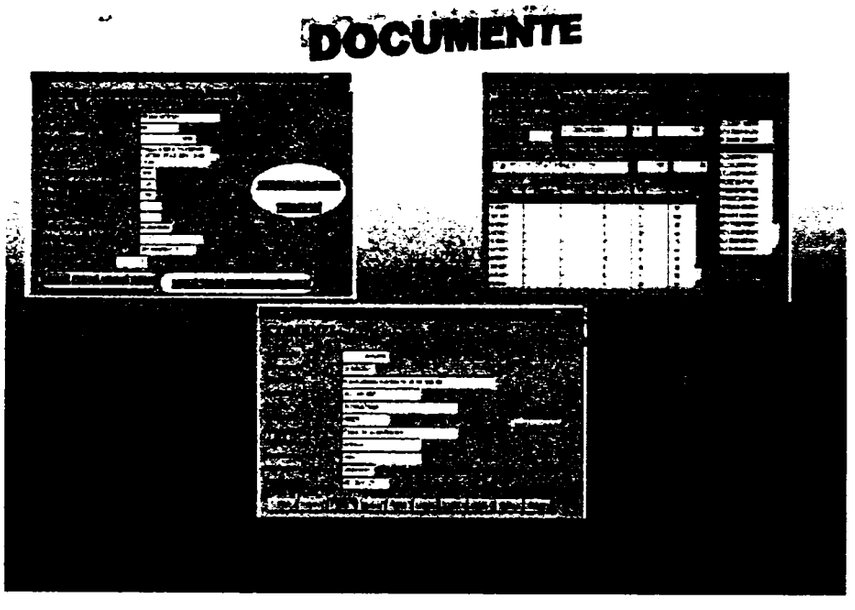


Fig 8.60 Clasa "Documente Economice"

Clasa "*Alezoare*" este prezentată în figura 8.61. De la obiectele buton, specifice clasei butoane, existente în clasă, pot fi apelate bonuri de consum (lansare), bon de materiale sau plan de operații, pe lângă activitățile enunțate anterior.

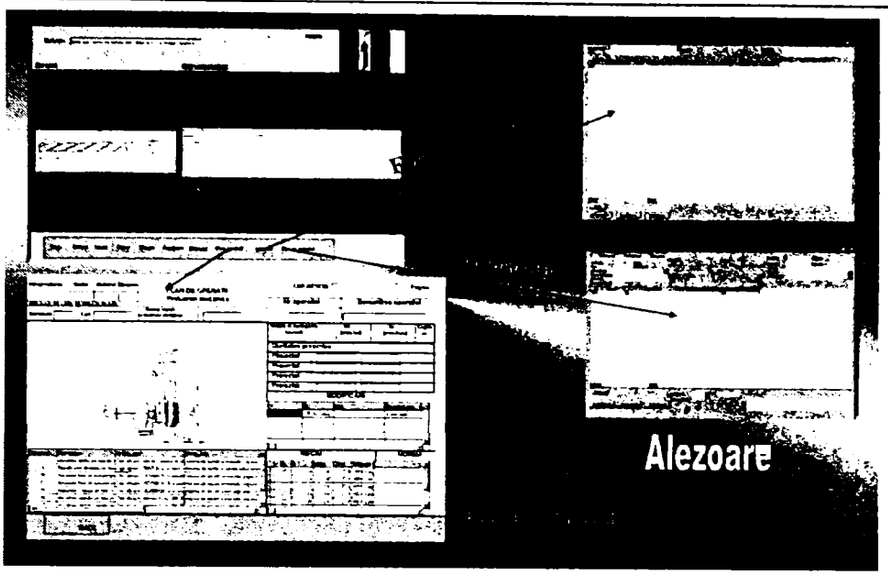


Fig 8. 61 Clasa "Alezoare"

Clasa *"Burghie"* este prezentată în figura 8.62. Fişa tehnologică de prelucrări este o clasă, care conţine butoane ce permit accesul la fişa de tratament termic sau la fişa tehnică de sudură, funcţie de dorinţa utilizatorului.

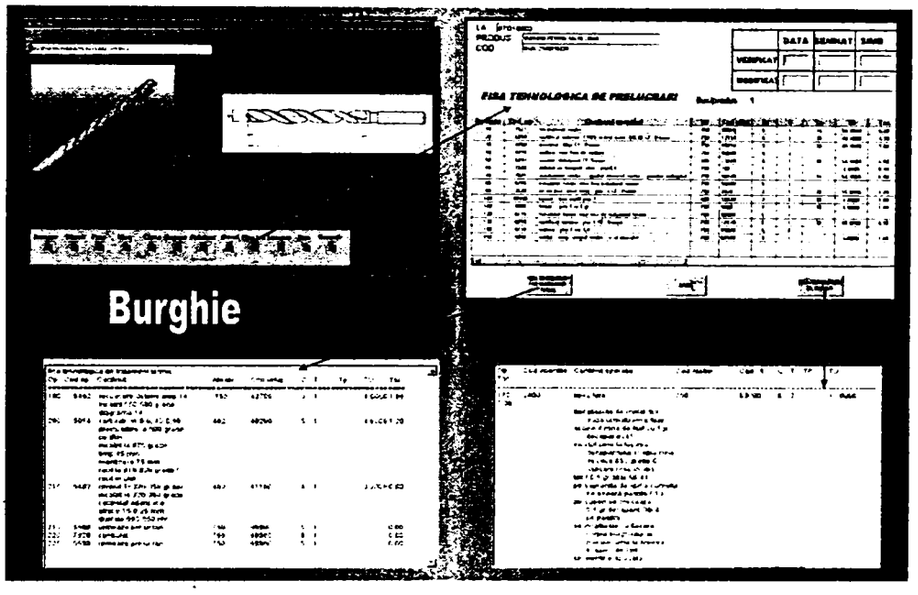


Fig 8.62 Clasa "Burghie"

Clasa "C*uţite"* este prezentată în figura 8.63. La fel ca şi la celelalte clase, este permisă vizualizarea fişei tehnologice, a reperelor pe produs sau a produselor fabricate în societatea economică.

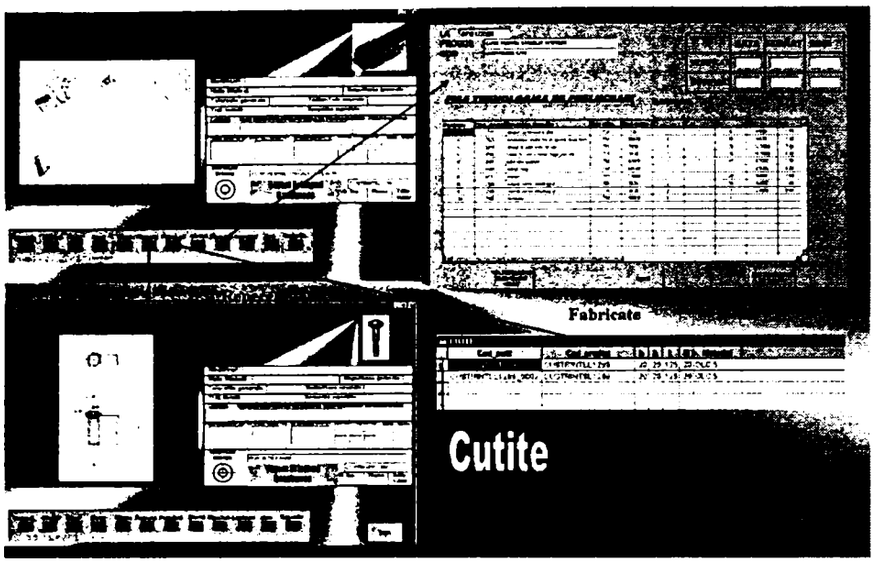


Fig 8.63 Clasa "Cuțite"

Prezentarea clasei *"Freze"* în figura 8.64, este axată pe posibilitățile de proiectare, pe care clasele aferente aplicației pentru SFS, le oferă. Sunt prezentate în mod succesiv mai multe clase, care permit introducerea unor date inițiale, pentru ca în final să se calculeze valorile mărimilor dorite, cu apariția grafică şi video a entității calculate. Clasele conțin obiecte, care permit introducerea mărimilor dorite, aceste obiecte fiind dotate cu metode care execută acțiunile dorite.

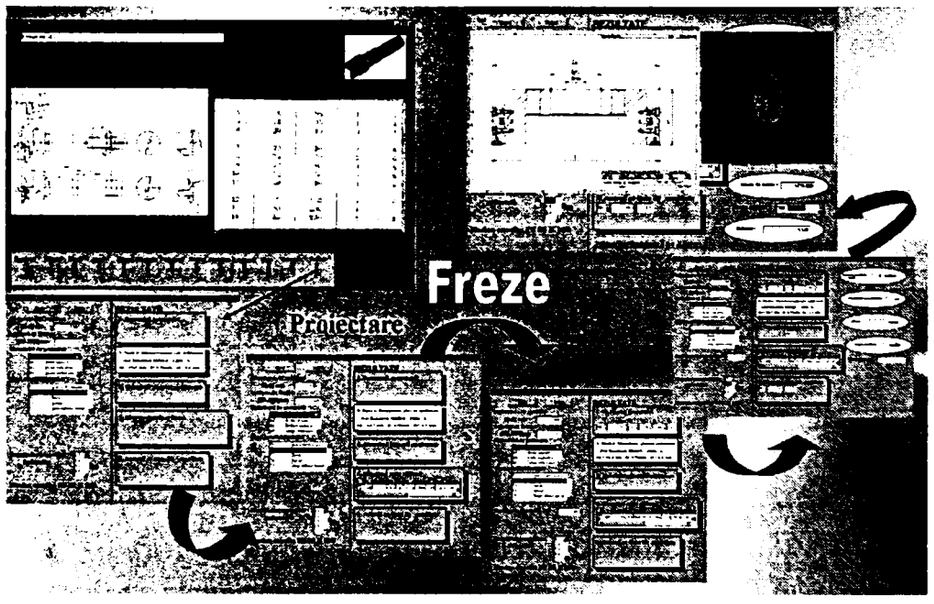


Fig 8.64 Clasa "Freze"

Clasa *"Şabloane"* (figura 8.65), permite vizualizarea unor clase de obiecte corespunzătoare documentelor economice, într-un mod diferit față de celelalte clase.

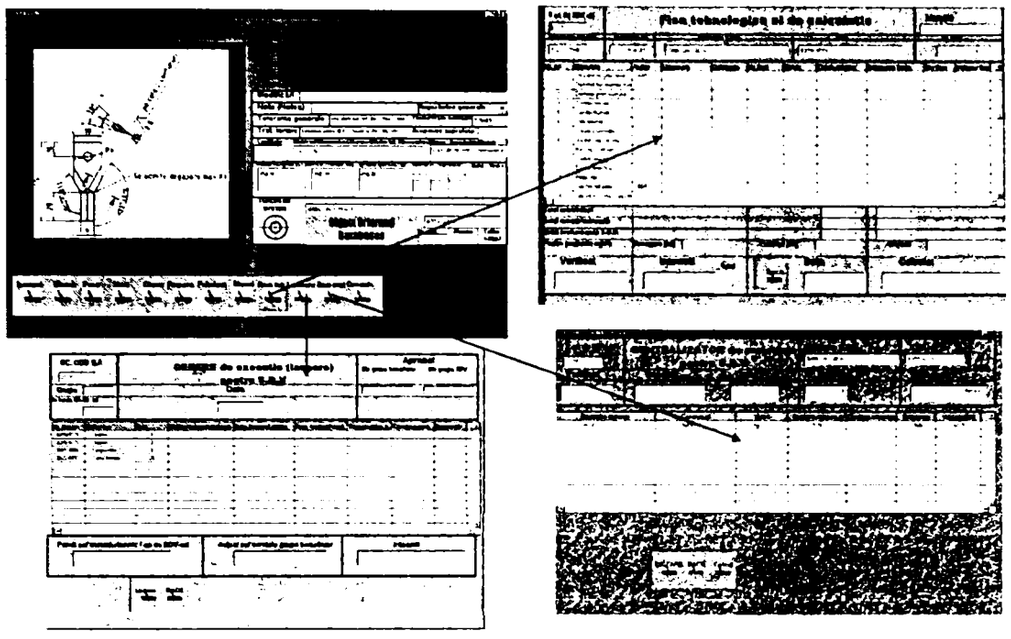


Fig 8.65 Clasa "Şabloane"

Structura ierarhică pentru clasele "Sabloane" este prezentată în figura 8.66.

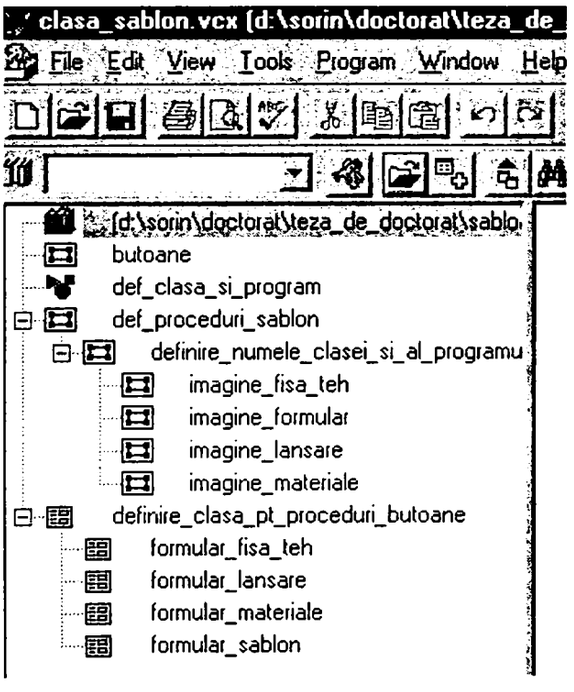


Fig 8. 66 Structura ierarhică pentru clasele "Sabloane"

Codul sursă al programului de lansare al ramurii aplicației "Şabloane" este următorul:

```
set talk off
set scoreboard off
close all
clea all
set classlib to clasa_sablon,flatbtn
```

```
oformsablon=createobject("formular_sablon")
oformfisatehs=createobject("formular_fisa_teh")
oformmaterial=createobject("formular_materiale")
oformlansare=createobject("formular_lansare")
oformsablon.show()
read events
```

Codul sursă al programului de generare a bazei de date obiectuale "Şablon" este:

```
set classlib to clasa_generare_clasa_de_date
odbcgen=createobject("creaza_clasa_de_date")
odbcgen.cdbcname="sablon.dbc"
odbcgen.cprgname="baza_de_date_sablon.prg"
odbcgen.doit()
```

Programul sursă generat de către programul precedent, pentru baza de date "Şablon" este:

```
*program.....: baza_de_date_sablon.prg
*baza de date.....: sablon.dbc
*generat in.....: April 06, 2002 - 14:25:26
#define databasepath ".F."
define class ARTICOL_SABLON as cursor
  alias = "ARTICOL_SABLON"
  cursorsource="ARTICOL_SABLON"
  database = "sablon.dbc"
enddefine
define class FISA_TEHNOLOG as cursor
  alias = "FISA_TEHNOLOG"
  cursorsource="FISA_TEHNOLOG"
  database = "sablon.dbc"
enddefine
define class COMANDA as cursor
  alias = "COMANDA"
  cursorsource="COMANDA"
  database = "sablon.dbc"
enddefine
define class EXECUTIE as cursor
  alias = "EXECUTIE"
  cursorsource="EXECUTIE"
  database = "sablon.dbc"
enddefine
define class MATERIALE as cursor
  alias = "MATERIALE"
  cursorsource="MATERIALE"
  database = "sablon.dbc"
enddefine
define class SABLON120 as cursor
  alias = "SABLON120"
  cursorsource="SABLON120"
  database = "sablon.dbc"
enddefine
define class SABLON75 as cursor
  alias = "SABLON75"
  cursorsource="SABLON75"
  database = "sablon.dbc"
enddefine
define class sablon as dataenvironment
add object oARTICOL_SABLON as ARTICOL_SABLON
add object oFISA_TEHNOLOG as FISA_TEHNOLOG
add object oCOMANDA as COMANDA
add object oEXECUTIE as EXECUTIE
add object oMATERIALE as MATERIALE
add object oSABLON120 as SABLON120
add object oSABLON75 as SABLON75
procedure vedef
public varsablon,varprodus
        select(this.initialselectedalias)
        brow noappe nodelete noedit
        varsablon=cods
        varprodus=codp
endproc
procedure alege
        select (this.initialselectedalias)
*       requery()
endproc
enddefine
```

Codul sursă al clasei care defineşte procedurile ramurii de program "Şablon" este:

```
DEFINE CLASS def_proceduri_sablon AS container
        Width = 200
        Height = 200
        PROTECTED ctablename
        ctablename = "articol_sablon"
        Name = "def_proceduri_sablon"
ADD OBJECT def_clasa_si_program1 AS def_clasa_si_program WITH ;
                Top = 12, ;
                Left = 12, ;
                Height = 48, ;
                Width = 48, ;
                Name = "Def_clasa_si_program1"
PROCEDURE inceput
        select (this.ctablename)
        go top
        thisform.refresh()
        if type ("thisform")="o"
                thisform.refresh()
        endif
```

```
        ENDPROC
PROCEDURE sfarsit
        select(this.ctablename)
        go bottom
                thisform.refresh()
        if type ("thisform")="o"
                thisform.refresh()
        endif
ENDPROC
PROCEDURE nextit
        select(this.ctablename)
        skip 1
        if eof()
                ??chr(7)
                wait window nowait "la sfarsit de fisier"
                go bottom
        endif
                thisform.refresh()
ENDPROC
PROCEDURE previt
        select (this.ctablename)
        skip -1
        if bof()
                ??chr(7)
                wait window nowait "la inceput de fisier"
                go top
        endif
        thisform.refresh()
ENDPROC
PROCEDURE closeit
        *release classlib flatbtn
        *release classlib clasa_sablon
        *release all
        *clear events
        *close all
        thisform.visible=.f.
ENDPROC
PROCEDURE fabricateit
        public varcutit,varprodusc
        do case
        case alltrim(oformsablon.imagine_formular1.text1.value)="SABLON 120+/-1"
                lodesab.initialselectedalias="sablon120"
                lodesab.vedef()
        case alltrim(oformsablon.imagine_formular1.text1.value)="SABLON 75"
                lodesab.initialselectedalias="sablon75"
                lodesab.vedef()
        endcase
ENDPROC
PROCEDURE repereit
        public vrepercutit, vcodprodcutit
        do case
        case alltrim(oformcutit.imagine_formular1.text1.value)="CUTIT PENTRU STRUNJIT INTERIOR"
                oformcutit.imagine_formular1.btnrevin.visible=.t.
        *       oformcutit.imagine_formular1.label3.visible=.f.
        *       oformcutit.imagine_formular1.oleboundcontrol3.visible=.f.
                lodesab.initialselectedalias="w_repere_cutit_str_int"
                lodesab.alege()
                oformcutit.imagine_formular1.text1.controlsource="w_repere_cutit_str_int.denumire"
                oformcutit.imagine_formular1.oleboundcontrol1.controlsource="w_repere_cutit_str_int.desen"
                oformcutit.imagine_formular1.oleboundcontrol2.controlsource="w_repere_cutit_str_int.imagine"
                oformcutit.imagine_formular1.ctablename="w_repere_cutit_str_int"
                thisform.refresh()
        case alltrim(oformcutit.imagine_formular1.text1.value)="CUTIT DE STRUNG CU PLACUTE DIN CARBURI METALICE"
                oformcutit.imagine_formular1.btnrevin.visible=.t.
        *       oformcutit.imagine_formular1.label3.visible=.f.
        *       oformcutit.imagine_formular1.oleboundcontrol3.visible=.f.
                lodesab.initialselectedalias="w_repere_cutit_strung_cu_placute"
                lodesab.alege()
                oformcutit.imagine_formular1.text1.controlsource="w_repere_cutit_strung_cu_placute.denumire"
                oformcutit.imagine_formular1.oleboundcontrol1.controlsource="w_repere_cutit_strung_cu_placute.desen"
                oformcutit.imagine_formular1.oleboundcontrol2.controlsource="w_repere_cutit_strung_cu_placute.imagine"
                oformcutit.imagine_formular1.ctablename="w_repere_cutit_strung_cu_placute"
```

```
                thisform.refresh()
        endcase
        ENDPROC
        PROCEDURE revinit
                oformcutit.imagine_formular1.text1.controlsource="articole_cutite_strung.denumire"
                oformcutit.imagine_formular1.oleboundcontrol1.controlsource="articole_cutite_strung.desen"
                oformcutit.imagine_formular1.oleboundcontrol2.controlsource="articole_cutite_strung.imagine"
                oformcutit.imagine_formular1.ctablename="articole_cutite_strung"
                oformcutit.imagine_formular1.btnrevin.visible=.f.
                oformcutit.imagine_formular1.label3.visible=.t.
        *       oformcutit.imagine_formular1.oleboundcontrol3.visible=.t.
                thisform.refresh()
        ENDPROC
        PROCEDURE fisa_tehnologicait
                do case
                case alltrim(oformsablon.imagine_formular1.text1.value)="SABLON 120+/-1"
                        lodesab.initialselectedalias="SABLON120"
                        lodesab.alege()
                        oformfisatehS.visible=.t.
                        thisform.refresh()
                        oformfisatehS.show()
                oformfisatehs.imagine_fisa_teh1.text14.value=0
                oformfisatehs.imagine_fisa_teh1.text14.refresh()
                oformfisatehs.imagine_fisa_teh1.text17.value=0
                oformfisatehs.imagine_fisa_teh1.text17.refresh()
                oformfisatehs.imagine_fisa_teh1.text18.value=0
                oformfisatehs.imagine_fisa_teh1.text18.refresh()
                oformfisatehs.imagine_fisa_teh1.text4.value="SABLON 120 +/-1"
                oformfisatehs.imagine_fisa_teh1.text5.value="8370-0774"
                oformfisatehs.imagine_fisa_teh1.text15.value=0
                oformfisatehs.imagine_fisa_teh1.text15.refresh()
                oformfisatehs.imagine_fisa_teh1.text16.value=0
                oformfisatehs.imagine_fisa_teh1.text16.refresh()
                oformfisatehs.imagine_fisa_teh1.text19.value=0
                oformfisatehs.imagine_fisa_teh1.text19.refresh()
                thisform.refresh()
                case alltrim(oformsablon.imagine_formular1.text1.value)="SABLON 75"
                        lodesab.initialselectedalias="SABLON75"
                        lodesab.alege()
                        oformfisatehS.visible=.t.
                        oformfisatehS.show()
                oformfisatehs.imagine_fisa_teh1.text14.value=0
                oformfisatehs.imagine_fisa_teh1.text14.refresh()
                oformfisatehs.imagine_fisa_teh1.text17.value=0
                oformfisatehs.imagine_fisa_teh1.text17.refresh()
                oformfisatehs.imagine_fisa_teh1.text18.value=0
                oformfisatehs.imagine_fisa_teh1.text18.refresh()
                oformfisatehs.imagine_fisa_teh1.text4.value="SABLON 75"
                oformfisatehs.imagine_fisa_teh1.text5.value="8370-0775"
                oformfisatehs.imagine_fisa_teh1.text15.value=0
                oformfisatehs.imagine_fisa_teh1.text15.refresh()
                oformfisatehs.imagine_fisa_teh1.text16.value=0
                oformfisatehs.imagine_fisa_teh1.text16.refresh()
                oformfisatehs.imagine_fisa_teh1.text19.value=0
                oformfisatehs.imagine_fisa_teh1.text19.refresh()
                thisform.refresh()
                endcase
        ENDPROC
        PROCEDURE Init
                select(this.ctablename)
                if type ("thisform.cclass")#"U"
                        thisform.cclass=this.name
                endif
        ENDPROC
ENDDEFINE
```

Codul sursă al clasei fişa tehnologică este următorul:

```
DEFINE        CLASS        formular_fisa_teh        AS
definire_clasa_pt_proceduri_butoane
        Top = -7
        Left = 2
        Height = 742
        Width = 1005
        ScrollBars = 0
        DoCreate = .T.
        WindowState = 2
        Name = "formular_fisa_teh"
        Butoane1.Timer_flat1.Name = "Timer_flat1"
```

Butoane1.butclose.Shadow.Name = "Shadow"
Butoane1.butclose.Line4.DefLeft = ""
Butoane1.butclose.Line4.DefHeight = ""
Butoane1.butclose.Line4.Name = "Line4"
Butoane1.butclose.Line2.DefWidth = ""
Butoane1.butclose.Line2.Name = "Line2"
Butoane1.butclose.Line1.DefTop = ""
Butoane1.butclose.Line1.DefWidth = ""
Butoane1.butclose.Line1.Name = "Line1"
Butoane1.butclose.Line3.DefHeight = ""
Butoane1.butclose.Line3.Name = "Line3"
Butoane1.butclose.Shape.Name = "Shape"
Butoane1.butclose.Imageoff.Name = "Imageoff"
Butoane1.butclose.Image.Name = "Image"
Butoane1.butclose.Label.Name = "Label"
Butoane1.butclose.ZOrderSet = 1
Butoane1.butclose.Name = "butclose"
Butoane1.buturmator.Shadow.Name = "Shadow"
Butoane1.buturmator.Line4.DefLeft = ""
Butoane1.buturmator.Line4.DefHeight = ""
Butoane1.buturmator.Line4.Name = "Line4"
Butoane1.buturmator.Line2.DefWidth = ""
Butoane1.buturmator.Line2.Name = "Line2"
Butoane1.buturmator.Line1.DefTop = ""
Butoane1.buturmator.Line1.DefWidth = ""
Butoane1.buturmator.Line1.Name = "Line1"
Butoane1.buturmator.Line3.DefHeight = ""
Butoane1.buturmator.Line3.Name = "Line3"
Butoane1.buturmator.Shape.Name = "Shape"
Butoane1.buturmator.Imageoff.Name = "Imageoff"
Butoane1.buturmator.Image.Name = "Image"
Butoane1.buturmator.Label.Name = "Label"
Butoane1.buturmator.ZOrderSet = 2
Butoane1.buturmator.Name = "buturmator"
Butoane1.butpreced.Shadow.Name = "Shadow"
Butoane1.butpreced.Line4.DefLeft = ""
Butoane1.butpreced.Line4.DefHeight = ""
Butoane1.butpreced.Line4.Name = "Line4"
Butoane1.butpreced.Line2.DefWidth = ""
Butoane1.butpreced.Line2.Name = "Line2"
Butoane1.butpreced.Line1.DefTop = ""
Butoane1.butpreced.Line1.DefWidth = ""
Butoane1.butpreced.Line1.Name = "Line1"
Butoane1.butpreced.Line3.DefHeight = ""
Butoane1.butpreced.Line3.Name = "Line3"
Butoane1.butpreced.Shape.Name = "Shape"
Butoane1.butpreced.Imageoff.Name = "Imageoff"
Butoane1.butpreced.Image.Name = "Image"
Butoane1.butpreced.Label.Name = "Label"
Butoane1.butpreced.ZOrderSet = 3
Butoane1.butpreced.Name = "butpreced"
Butoane1.butsfarsit.Shadow.Name = "Shadow"
Butoane1.butsfarsit.Line4.DefLeft = ""
Butoane1.butsfarsit.Line4.DefHeight = ""
Butoane1.butsfarsit.Line4.Name = "Line4"
Butoane1.butsfarsit.Line2.DefWidth = ""
Butoane1.butsfarsit.Line2.Name = "Line2"
Butoane1.butsfarsit.Line1.DefTop = ""
Butoane1.butsfarsit.Line1.DefWidth = ""
Butoane1.butsfarsit.Line1.Name = "Line1"
Butoane1.butsfarsit.Line3.DefHeight = ""
Butoane1.butsfarsit.Line3.Name = "Line3"
Butoane1.butsfarsit.Shape.Name = "Shape"
Butoane1.butsfarsit.Imageoff.Name = "Imageoff"
Butoane1.butsfarsit.Image.Name = "Image"
Butoane1.butsfarsit.Label.Name = "Label"
Butoane1.butsfarsit.ZOrderSet = 4
Butoane1.butsfarsit.Name = "butsfarsit"
Butoane1.butinceput.Shadow.Name = "Shadow"
Butoane1.butinceput.Line4.DefLeft = ""
Butoane1.butinceput.Line4.DefHeight = ""
Butoane1.butinceput.Line4.Name = "Line4"
Butoane1.butinceput.Line2.DefWidth = ""
Butoane1.butinceput.Line2.Name = "Line2"
Butoane1.butinceput.Line1.DefTop = ""
Butoane1.butinceput.Line1.DefWidth = ""
Butoane1.butinceput.Line1.Name = "Line1"
Butoane1.butinceput.Line3.DefHeight = ""
Butoane1.butinceput.Line3.Name = "Line3"
Butoane1.butinceput.Shape.Name = "Shape"
Butoane1.butinceput.Imageoff.Name = "Imageoff"
Butoane1.butinceput.Image.Name = "Image"
Butoane1.butinceput.Label.FontSize = 12
Butoane1.butinceput.Label.Caption = "Back"
Butoane1.butinceput.Label.Left = 12
Butoane1.butinceput.Label.Top = 12
Butoane1.butinceput.Label.Name = "Label"
Butoane1.butinceput.ZOrderSet = 12
Butoane1.butinceput.Name = "butinceput"
Butoane1.btnreper.Shadow.Name = "Shadow"
Butoane1.btnreper.Line4.DefLeft = ""
Butoane1.btnreper.Line4.DefHeight = ""
Butoane1.btnreper.Line4.Name = "Line4"
Butoane1.btnreper.Line2.DefWidth = ""
Butoane1.btnreper.Line2.Name = "Line2"
Butoane1.btnreper.Line1.DefTop = ""
Butoane1.btnreper.Line1.DefWidth = ""
Butoane1.btnreper.Line1.Name = "Line1"
Butoane1.btnreper.Line3.DefHeight = ""
Butoane1.btnreper.Line3.Name = "Line3"
Butoane1.btnreper.Shape.Name = "Shape"
Butoane1.btnreper.Imageoff.Name = "Imageoff"
Butoane1.btnreper.Image.Name = "Image"
Butoane1.btnreper.Label.Name = "Label"
Butoane1.btnreper.ZOrderSet = 5
Butoane1.btnreper.Name = "btnreper"
Butoane1.btnfisa.Shadow.Name = "Shadow"
Butoane1.btnfisa.Line4.DefLeft = ""
Butoane1.btnfisa.Line4.DefHeight = ""
Butoane1.btnfisa.Line4.Name = "Line4"
Butoane1.btnfisa.Line2.DefWidth = ""
Butoane1.btnfisa.Line2.Name = "Line2"
Butoane1.btnfisa.Line1.DefTop = ""
Butoane1.btnfisa.Line1.DefWidth = ""
Butoane1.btnfisa.Line1.Name = "Line1"
Butoane1.btnfisa.Line3.DefHeight = ""
Butoane1.btnfisa.Line3.Name = "Line3"
Butoane1.btnfisa.Shape.Name = "Shape"
Butoane1.btnfisa.Imageoff.Name = "Imageoff"
Butoane1.btnfisa.Image.Name = "Image"
Butoane1.btnfisa.Label.Name = "Label"
Butoane1.btnfisa.ZOrderSet = 6
Butoane1.btnfisa.Name = "btnfisa"
Butoane1.btnlansare.Shadow.Name = "Shadow"
Butoane1.btnlansare.Line4.DefLeft = ""
Butoane1.btnlansare.Line4.DefHeight = ""
Butoane1.btnlansare.Line4.Name = "Line4"
Butoane1.btnlansare.Line2.DefWidth = ""
Butoane1.btnlansare.Line2.Name = "Line2"
Butoane1.btnlansare.Line1.DefTop = ""
Butoane1.btnlansare.Line1.DefWidth = ""
Butoane1.btnlansare.Line1.Name = "Line1"
Butoane1.btnlansare.Line3.DefHeight = ""
Butoane1.btnlansare.Line3.Name = "Line3"
Butoane1.btnlansare.Shape.Name = "Shape"
Butoane1.btnlansare.Imageoff.Name = "Imageoff"
Butoane1.btnlansare.Image.Name = "Image"
Butoane1.btnlansare.Label.Name = "Label"
Butoane1.btnlansare.ZOrderSet = 7
Butoane1.btnlansare.Name = "btnlansare"
Butoane1.btnbonmat.Shadow.Name = "Shadow"
Butoane1.btnbonmat.Line4.DefLeft = ""
Butoane1.btnbonmat.Line4.DefHeight = ""
Butoane1.btnbonmat.Line4.Name = "Line4"

```
Butoane1.btnbonmat.Line2.DefWidth = ""
Butoane1.btnbonmat.Line2.Name = "Line2"
Butoane1.btnbonmat.Line1.DefTop = ""
Butoane1.btnbonmat.Line1.DefWidth = ""
Butoane1.btnbonmat.Line1.Name = "Line1"
Butoane1.btnbonmat.Line3.DefHeight = ""
Butoane1.btnbonmat.Line3.Name = "Line3"
Butoane1.btnbonmat.Shape.Name = "Shape"
Butoane1.btnbonmat.Imageoff.Name = "Imageoff"
Butoane1.btnbonmat.Image.Name = "Image"
Butoane1.btnbonmat.Label.Name = "Label"
Butoane1.btnbonmat.ZOrderSet = 8
Butoane1.btnbonmat.Name = "btnbonmat"
Butoane1.btncomanda.Shadow.Name = "Shadow"
Butoane1.btncomanda.Line4.DefLeft = ""
Butoane1.btncomanda.Line4.DefHeight = ""
Butoane1.btncomanda.Line4.Name = "Line4"
Butoane1.btncomanda.Line2.DefWidth = ""
Butoane1.btncomanda.Line2.Name = "Line2"
Butoane1.btncomanda.Line1.DefTop = ""
Butoane1.btncomanda.Line1.DefWidth = ""
Butoane1.btncomanda.Line1.Name = "Line1"
Butoane1.btncomanda.Line3.DefHeight = ""
Butoane1.btncomanda.Line3.Name = "Line3"
Butoane1.btncomanda.Shape.Name = "Shape"
Butoane1.btncomanda.Imageoff.Name = "Imageoff"
Butoane1.btncomanda.Image.Name = "Image"
Butoane1.btncomanda.Label.Name = "Label"
Butoane1.btncomanda.ZOrderSet = 9
Butoane1.btncomanda.Name = "btncomanda"
Butoane1.btnfabricat.Shadow.Name = "Shadow"
Butoane1.btnfabricat.Line4.DefLeft = ""
Butoane1.btnfabricat.Line4.DefHeight = ""
Butoane1.btnfabricat.Line4.Name = "Line4"
Butoane1.btnfabricat.Line2.DefWidth = ""
Butoane1.btnfabricat.Line2.Name = "Line2"
Butoane1.btnfabricat.Line1.DefTop = ""
Butoane1.btnfabricat.Line1.DefWidth = ""
Butoane1.btnfabricat.Line1.Name = "Line1"
Butoane1.btnfabricat.Line3.DefHeight = ""
Butoane1.btnfabricat.Line3.Name = "Line3"
Butoane1.btnfabricat.Shape.Name = "Shape"
Butoane1.btnfabricat.Imageoff.Name = "Imageoff"
Butoane1.btnfabricat.Image.Name = "Image"
Butoane1.btnfabricat.Label.Name = "Label"
Butoane1.btnfabricat.ZOrderSet = 10
Butoane1.btnfabricat.Name = "btnfabricat"
Butoane1.btnstoc.Shadow.Name = "Shadow"
Butoane1.btnstoc.Line4.DefLeft = ""
Butoane1.btnstoc.Line4.DefHeight = ""
Butoane1.btnstoc.Line4.Name = "Line4"
Butoane1.btnstoc.Line2.DefWidth = ""
Butoane1.btnstoc.Line2.Name = "Line2"
Butoane1.btnstoc.Line1.DefTop = ""
Butoane1.btnstoc.Line1.DefWidth = ""
Butoane1.btnstoc.Line1.Name = "Line1"
Butoane1.btnstoc.Line3.DefHeight = ""
Butoane1.btnstoc.Line3.Name = "Line3"
Butoane1.btnstoc.Shape.Name = "Shape"
Butoane1.btnstoc.Imageoff.Name = "Imageoff"
Butoane1.btnstoc.Image.Name = "Image"
Butoane1.btnstoc.Label.Name = "Label"
Butoane1.btnstoc.ZOrderSet = 11
Butoane1.btnstoc.Name = "btnstoc"
Butoane1.Top = 540
Butoane1.Left = 468
Butoane1.Width = 84
Butoane1.Height = 75
Butoane1.ZOrderSet = 3
Butoane1.Name = "Butoane1"
ADD OBJECT timer_flat1 AS timer_flat WITH ;
                Top = 612, ;
                Left = 924, ;
                Height = 24, ;
                Width = 24, ;
                Name = "Timer_flat1"
ADD OBJECT imagine_fisa_teh1 AS imagine_fisa_teh WITH ;
                Top = 0, ;
                Left = 12, ;
                Width = 1008, ;
                Height = 744, ;
                ZOrderSet = 1, ;
                Name = "Imagine_fisa_teh1", ;
                Shape1.Name = "Shape1", ;
Def_clasa_si_program1.Name = "Def_clasa_si_program1", ;
                Label6.Name = "Label6", ;
                Label5.Name = "Label5", ;
                Label4.Name = "Label4", ;
                Label3.Name = "Label3", ;
                Label2.Name = "Label2", ;
                Label1.Name = "Label1", ;
                Label9.Left = 0, ;
                Label9.Top = 432, ;
                Label9.Name = "Label9", ;
                Label10.Left = 0, ;
                Label10.Top = 456, ;
                Label10.Name = "Label10", ;
                Label11.Left = 0, ;
                Label11.Top = 480, ;
                Label11.Name = "Label11", ;
                Label12.Left = 0, ;
                Label12.Top = 504, ;
                Label12.Name = "Label12", ;
                Label13.Left = 36, ;
                Label13.Top = 540, ;
                Label13.Name = "Label13", ;
                Label14.Left = 252, ;
                Label14.Top = 540, ;
                Label14.Name = "Label14", ;
                Label15.Left = 564, ;
                Label15.Top = 540, ;
                Label15.Name = "Label15", ;
                Label16.Left = 768, ;
                Label16.Top = 540, ;
                Label16.Name = "Label16", ;
                Label17.Left = 192, ;
                Label17.Top = 504, ;
                Label17.Name = "Label17", ;
                Label18.Left = 468, ;
                Label18.Top = 504, ;
                Label18.Name = "Label18", ;
                Label19.Left = 744, ;
                Label19.Top = 504, ;
                Label19.Name = "Label19", ;
                Text1.Name = "Text1", ;
                Text2.Name = "Text2", ;
                Text3.Name = "Text3", ;
                Text4.Name = "Text4", ;
                Text5.Name = "Text5", ;
                Text6.Name = "Text6", ;
                Line1.Name = "Line1", ;
                Line2.Name = "Line2", ;
                Line3.Name = "Line3", ;
                Label7.Name = "Label7", ;
                Line4.Name = "Line4", ;
                Line5.Name = "Line5", ;
                Line6.Name = "Line6", ;
                Label8.Name = "Label8", ;
        Grid1.Column1.Header1.Name = "Header1", ;
        Grid1.Column1.Text1.Name = "Text1", ;
        Grid1.Column1.Name = "Column1", ;
        Grid1.Column2.Header1.Name = "Header1", ;
        Grid1.Column2.Text1.Name = "Text1", ;
        Grid1.Column2.Name = "Column2", ;
        Grid1.Column3.Header1.Name = "Header1", ;
        Grid1.Column3.Text1.Name = "Text1", ;
```

```
Grid1.Column3.Name = "Column3", ;
Grid1.Column4.Header1.Name = "Header1", ;
Grid1.Column4.Text1.Name = "Text1", ;
Grid1.Column4.Name = "Column4", ;
Grid1.Column5.Header1.Name = "Header1", ;
Grid1.Column5.Text1.Name = "Text1", ;
Grid1.Column5.Name = "Column5", ;
Grid1.Column6.Header1.Name = "Header1", ;
Grid1.Column6.Text1.Name = "Text1", ;
Grid1.Column6.Name = "Column6", ;
Grid1.Column7.Header1.Name = "Header1", ;
Grid1.Column7.Text1.Name = "Text1", ;
Grid1.Column7.Name = "Column7", ;
Grid1.Column8.Header1.Name = "Header1", ;
Grid1.Column8.Text1.Name = "Text1", ;
Grid1.Column8.Name = "Column8", ;
Grid1.Column9.Header1.Name = "Header1", ;
Grid1.Column9.Text1.Name = "Text1", ;
Grid1.Column9.Name = "Column9", ;
Grid1.Column10.Header1.Name = "Header1", ;
Grid1.Column10.Text1.Name = "Text1", ;
Grid1.Column10.Name = "Column10", ;
Grid1.Column11.Header1.Name = "Header1", ;
Grid1.Column11.Text1.Name = "Text1", ;
Grid1.Column11.Name = "Column11", ;
Grid1.Height = 312, ;
Grid1.Left = 0, ;
Grid1.Top = 120, ;
Grid1.Width = 996, ;
Grid1.Name = "Grid1", ;
Line7.Left = -12, ;
Line7.Top = 456, ;
Line7.Name = "Line7", ;
Line8.Left = 0, ;
Line8.Top = 480, ;
Line8.Name = "Line8", ;
Line9.Left = 0, ;
Line9.Top = 504, ;
Line9.Name = "Line9", ;
Line10.Left = 0, ;
Line10.Top = 540, ;
Line10.Name = "Line10", ;
Line11.Left = 0, ;
Line11.Top = 624, ;
Line11.Name = "Line11", ;
Line12.Left = 192, ;
Line12.Top = 432, ;
Line12.Name = "Line12", ;
Line13.Left = 440, ;
Line13.Top = 432, ;
Line13.Name = "Line13", ;
Text7.Left = 300, ;
Text7.Top = 504, ;
Text7.Name = "Text7", ;
Text8.Left = 588, ;
Text8.Top = 504, ;
Text8.Name = "Text8", ;
Text9.Left = 828, ;
Text9.Top = 504, ;
Text9.Name = "Text9", ;
Text10.Left = 24, ;
Text10.Top = 576, ;
Text10.Name = "Text10", ;
Text11.Left = 216, ;
Text11.Top = 576, ;
Text11.Name = "Text11", ;
Text12.Height = 37, ;
Text12.Left = 564, ;
        Text12.Top = 576, ;
        Text12.Width = 132, ;
        Text12.Name = "Text12", ;
        Text13.Left = 732, ;
        Text13.Top = 576, ;
        Text13.Name = "Text13", ;
        Text14.Left = 600, ;
        Text14.Top = 432, ;
        Text14.Name = "Text14", ;
        Text15.Left = 720, ;
        Text15.Top = 432, ;
        Text15.Name = "Text15", ;
        Text16.Left = 720, ;
        Text16.Top = 456, ;
        Text16.Name = "Text16", ;
        Line14.Left = 699, ;
        Line14.Top = 432, ;
        Line14.Name = "Line14", ;
        Text17.Left = 600, ;
        Text17.Top = 456, ;
        Text17.Name = "Text17", ;
        Text18.Left = 600, ;
        Text18.Top = 480, ;
        Text18.Name = "Text18", ;
        Text19.Left = 720, ;
        Text19.Top = 480, ;
        Text19.Name = "Text19", ;
        Text20.Height = 21, ;
        Text20.Top = 38, ;
        Text20.Name = "Text20"
    ADD OBJECT labelbtn1 AS labelbtn WITH ;
        Top = 552, ;
        Left = 384, ;
        Width = 60, ;
        Height = 24, ;
        ZOrderSet = 2, ;
        Name = "Labelbtn1", ;
        Shadow.Name = "Shadow", ;
        Line4.DefLeft = "", ;
        Line4.DefHeight = "", ;
        Line4.Name = "Line4", ;
        Line2.DefWidth = "", ;
        Line2.Name = "Line2", ;
        Line1.DefTop = "", ;
        Line1.DefWidth = "", ;
        Line1.Name = "Line1", ;
        Line3.DefHeight = "", ;
        Line3.Name = "Line3", ;
        Shape.Name = "Shape", ;
        Imageoff.Name = "Imageoff", ;
        Image.Name = "Image", ;
        Label.Caption = "Calcul", ;
        Label.Name = "Label"
    PROCEDURE Butoane1.butinceput.Click
        thisform.visible=.f.
    ENDPROC
    PROCEDURE labelbtn1.Click
    lodesab.initialselectedalias="fisa_tehnolog"
        lodesab.alege()
        go top
    oformfisatehs.imagine_fisa_teh1.grid1.refresh()
    sum(fisa_tehnolog.colabi) to a
    oformfisatehs.imagine_fisa_teh1.text14.value=int(a/c
omanda.nr_buc)
    oformfisatehs.imagine_fisa_teh1.text14 refresh()
    oformfisatehs.imagine_fisa_teh1.text17.value=a
    oformfisatehs.imagine_fisa_teh1.text17.refresh()
oformfisatehs.imagine_fisa_teh1.text18.value=int(a/60)
    oformfisatehs.imagine_fisa_teh1.text18.refresh()
    sum(fisa_tehnolog.echipa) to a
oformfisatehs.imagine_fisa_teh1.text15.value=int(a/comanda.nr
_buc)
    oformfisatehs.imagine_fisa_teh1.text15.refresh()
    oformfisatehs.imagine_fisa_teh1.text16.value=a
    oformfisatehs.imagine_fisa_teh1.text16.refresh()
oformfisatehs.imagine_fisa_teh1.text19.value=int(a/60)
```

```
oformfisatehs imagine_fisa_teh1 text19.refresh()          ENDDEFINE
clea
ENDPROC
```

Celelalte clase sunt asemănătoare cu cele prezentate anterior.

Pentru fiecare categorie de scule, conform clasificării existente există posibilitatea vizualizării unor scule STAS. Clasa stas conține un meniu obiect, de la care poate fi apelat stas-ul dorit (figura 8.67).

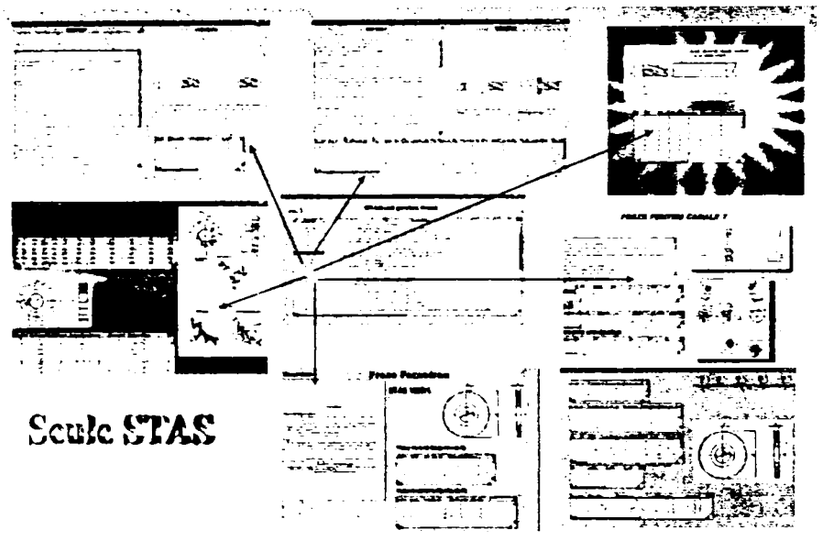


Fig 8.67 Clasa "Stas"

În cazul care se dorește o sculă specială, indiferent de clasa acestora, poate fi apelat un catalog de firmă, fie de pe internet, fie dintr-o bază de date obiectuală, care memorează cataloagele diferitelor firme producătoare de scule (figura 8.68).

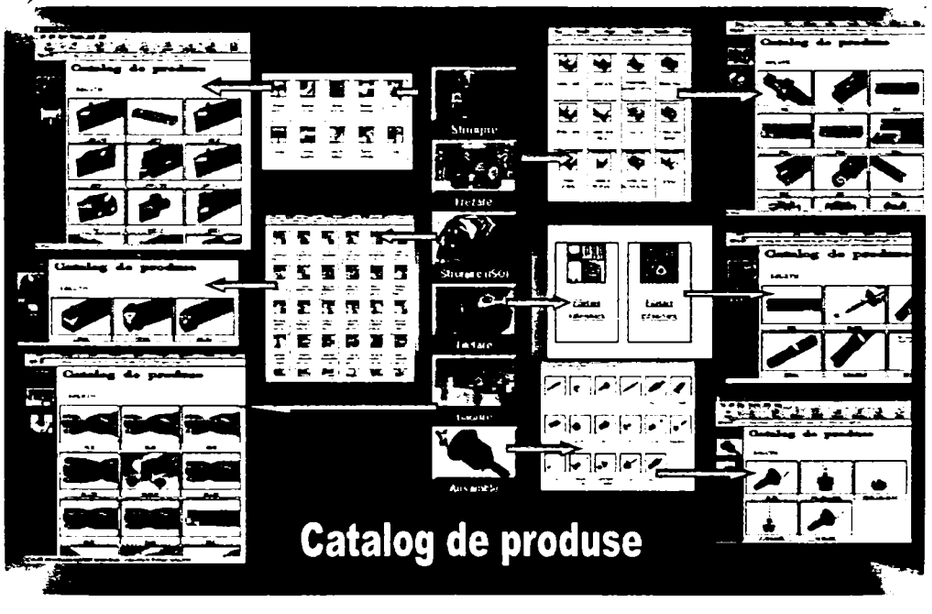


Fig 8.68 Clasa "Catalog Produse"

În implementarea obiectual relațională realizată, sunt de subliniat câteva aspecte:

1. într-un mediu relațional obiectual, pornind de la suportul de memorare relațional, s-a dovedit posibilitatea realizării unei implementări pur obiectuale, în care atât mediul de memorare, cât și programele de gestionare a obiectelor sunt obiectuale;
2. generarea bazelor de date obiectuale, pornind de la cele relaționale s-a făcut prin program, folosind tehnologia obiectuală;
3. metodologia de generare a schemelor obiectuale și a claselor derivate a fost demonstrată și în mediul relațional obiectual, fiind prin aceasta o extindere a standardului ODMG 3.0 care se referă la bazele de date obiectuale;
4. din punct de vedere al sistemului de fabricație al sculelor, aplicația realizată este un lucru nou, implementarea Sistemului de Fabricație al Sculelor, într-un mediu relațional obiectual a scos în evidență avantajele folosiri unui astfel de mediu, pentru aplicații de gen CAD/CAM.

9

Concluzii şi Perspective

În această teză s-a realizat o bază de date orientată obiect pentru Sistemul de Fabricaţie al Sculelor, implementată atât într-un mediu pur obiectual (CA Jasmine) cât şi într-un mediu hibrid, relaţional obiectual (Micrsoft Visual Fox). În teză, s-a făcut o analiză detaliată a principalelor concepte obiectuale aplicate bazelor de date orientate obiect, a modului de implementare şi realizare a acestor. În acelaşi timp s-a elaborat o nouă metodologie de definire a schemelor externe, bazată pe arhitectura ANSI/SPARC pe trei nivele. Referitor la acest aspect, teza aduce noutăţi privind modul de interpretare şi de rezolvare a problemelor, pe baza conceptelor obiectuale fundamentale. Implementarea conceptelor teoretice referitoare la schemele externe şi la clasele derivate, atât într-un mediu pur obiectual cât şi într-un mediu relaţional obiectual, este un domeniu în care teza, aduce importante contribuţii. Soluţia aleasă pentru Sistemul de Fabricaţie al Sculelor, precum şi modul de implementare şi realizare sunt complet noi. Contribuţiile pe care le aduce teza sunt:

I. Se face o analiză detaliată a tipurilor de baze de date obiectuale existente, a modului în care principalele concepte fundamentale sunt aplicate în domeniul bazelor de date. Se constată existenţa a două tipuri fundamentale de baze de date orientate obiect:

   A. Cele pur obiectuale, la care atât modelul de date, cât şi limbajul de programare folosit pentru dezvoltarea aplicaţiilor este obiectual. Obiectele pot fi memorate în baza de date, li se pot modifica proprietăţile, este asigurată moştenirea, încapsularea, polimorfismul, identitatea obiectuală. Toate aceste caracteristici au fost reliefate în implementarea făcută în mediul Jasmine de la Computer Associates;
   B. Cele hibride sau relaţional obiectuale, în care mediul de memorare este relaţional, iar limbajul de programare este obiectual. Se constată în versiunile mai noi ale ORDB-urilor, definirea conceptului de clasă atât pentru mediile de stocare. Acest lucru a fost subliniat în capitolul 8, în care implementarea făcută în Visual Fox, se apropie mai mult, de un mediu pur obiectual, bazele de date obiectuale fiind generate de alte clase de obiecte, cu toate că se porneşte de la baza de date relaţională.

II. La momentul actual, în domeniul bazelor de date pe plan mondial, se observă o conectare puternică intre relațional şi obiectual. Teza scoate în evidență aceste aspecte, prin modul de tratare teoretic, pornind de la o arhitectură tradițional relațională, aplicată pe un mediu pur obiectual şi pe un mediu hibrid. Aplicabilitatea pe mediul hibrid s-a făcut datorită faptului că aceste medii soft sunt cele mai răspândite la momentul actual. Avantajele unei astfel de soluții au fost prezentate în teză: uşurința definirii obiectelor de date, folosirea avantajelor mediilor obiectuale, clase, clase derivate, folosirea mediilor relaționale pentru stocarea datelor, folosirea limbajului SQL pentru realizarea interogărilor şi a vederilor;

III. Pentru bazele de date orientate obiect, teza propune o nouă metodologie de definire a schemelor obiectuale Ea foloseşte concepte comune mediilor orientate obiect. Noua metodologie oferă o soluție pentru integrarea claselor derivate cu restul claselor existente. Integrarea este făcută în două faze: prima dată în dicționarul de date şi apoi în schema externă. Soluția propusă reduce numărul de clase auxiliare ce trebuie generate în scopul de a îndeplini integrarea claselor derivate, în comparație cu alte soluții. În acest sens este dezvoltat conceptul de clasă transformabilă sau netransformabilă. Acest concept este folosit în dezvoltarea algoritmior pentru generarea schemelor externe. Totodată a fost arătat modul cum acest concept, simplifică procesul de definire al schemei externe;

IV. Modul nou de studiere al identificatorilor de obiect prin folosirea semanticilor: obiect generat şi obiect conservat. În tot ceea ce se prezintă se face distincție clară între mecanismul de generare al identificatorului şi mecanismul de menținere a conexiunii între obiectele claselor de bază şi ale claselor derivate. Acest lucru oferă o soluție pentru transmiterea modificărilor între clasele de bază şi clasele derivate.

V. Clasele derivate oferă o nouă interfată pentru accesul la informația existentă în clasele de bază. În evoluția schemei obiectuale consider că este necesară definirea claselor parțial derivate, adică a claselor ce conțin elemente nederivabile. Dacă unele clase cu informație nederivabilă trebuie să fie incorporate în scheme externe, informația nederivabilă va fi inclusă în schema conceptuală. Unele operații de derivare prezentate în capitolul 5 pot fi folosite pentru a produce clase parțial derivabile. În cele prezentate clasele parțial derivate pot conține elemente nederivabile atât în grad cât şi în extensie. Clasele parțial derivate sunt definite în scopul de a satisface cerințele utilizatorilor. Aceste clase conțin atât informație derivabilă, cât şi informație nederivabilă. Schemele externe pot conține clase derivate definite prin semanticile obiect conservat şi obiect generat.

VI. Pornind de la definiția clasică a claselor derivate, ca rezultat al unei interogări, în cadrul tezei s-a extins acest concept, prin clasă derivată

înțelegându-se mai mult, funcție de metoda de derivare aleasă, prin moştenire sau printr-o relație derivată;

VII. În scopul de a permite modificări neprevăzute a fost definit conceptul de schemă externă temporală. Când o schemă externă cu informații nederivate este definită, schema conceptuală va fi modificată în scopul de a include informație nederivată în noua schemă;

VIII. Toate conceptele teoretice redefinite şi dezvoltate sunt implementate în bazele de date realizate, atât cele pur obiectuale cât şi cele hibride. Realizarea practică a bazelor de date, în cele două medii obiectuale reprezintă o contribuție importantă.

IX. Se face o analiză amănunțită a modului de implementare a bazei de date într-un mediu pur obiectual, respectând noțiunile teoretice dezvoltate în cadrul tezei. Implementarea obiectuală prezintă o serie de particularități, atât din punct de vedere conceptual cât şi din modul de reprezentare al datelor, prin dispariția noțiunii de relație, atributele acesteia regăsindu-se printre proprietățile obiectelor. Concluziile obținute din acest mod de implementare, au determinat alegerea unui SGBDOR, datorită faptului că per ansamblu, acesta prezintă o serie de avantaje. Teza aduce contribuții importante, privind modul de realizare conceptual, pur obiectual al bazei de date pe un mediu relațional, aplicând noțiunile teoretice dezvoltate în cadrul tezei.

X. Modul în care au fost tratate şi interpretate interogările şi vederile. Acestea au ca rezultat obiecte în SGBDOR. Entitatea vedere obiect, există independent în cadrul bazelor de date ale Sistemul de Fabricație al Sculelor. Din punct de vedere al definiției o vedere este considerată o clasă derivată. Modul în care sunt tratate datele este pur obiectual. Pe fondul realizării aplicației, conceptele teoretice dezvoltate în cadrul tezei, referitoare la clasele derivate şi schemele externe, au fost implementate. S-au găsit soluții unice privind modul în care au fost definite clasele derivate şi schemele externe, un aspect interesant constituindu-l modul în care au fost găsite soluțiile de generare a claselor de memorare folosind cod program. A fost definită o clasă de obiecte, care are ca scop generarea codului program sursă pentru generarea bazei de date obiectuală, pornind de la baza de date relațională.

XI. S-a făcut o analiză privind modul de funcționare al aplicației în faza de implementare. S-a constatat ca prezența obiectelor relații, a vederilor obiect (claselor derivate), întârzie lansarea în execuție a aplicației până în momentul în care aceste sunt executate în totalitate. O dată aplicația lansată în execuție viteza de lucru este comparabilă cu execuția lucrării într-un mediu relațional. Implementarea pur obiectuală a generat întârzieri de execuție comparativ cu mediul relațional obiectual. Aceste aspecte sunt influențate de mediul pur obiectual folosit (Jasmine). S-au constatat diferențe mari între abordarea clasică şi abordarea obiectuală a codului program folosit la implementarea bazei de date.

XII. Clasificarea sculelor este un lucru original, propriu acestui Sistem de Fabricație al Sculelor. Tratarea Sistemul de Fabricație al Sculelor, pe un sistem de baze de date orientate obiect este un lucru absolut nou. În general în Romania există puține aplicații care tratează acest aspect, şi cu atât mai puține cele care folosesc un mediu obiectual. Folosirea unui mediu obiectual prezintă o serie de avantaje nete din punct de vedere multimedia, desen imagine, video față de implementările clasice. Baza de date obiectuală face legătura cu medii CAD de proiectare, în mod concret CATIA. Modulul de proiectare exemplifică posibilitățile pe care le oferă baza de date orientată obiect în rezolvarea problemelor care apar în activitatea inginerilor tehnologi.

Noutățile pe care le aduce teza pot fi canalizate pe cele trei direcții importante: teoretic, al implementării, al Sistemul de Fabricație al Sculelor.

I. Se face o analiză originală a principalelor concepte referitoare la bazele de date orientate obiect, şi al bazelor de date relațional obiectuale, fundamentând soluția aleasă pentru realizarea Sistemul de Fabricație al Sculelor;

II. Sunt dezvoltate conceptele de "schemă externă" şi "clasă derivată", fiind propus conceptul nou de clasă derivată parțial, şi acela de schemă temporală. Se propune o metodă originală de generare a claselor derivate prin semantica "obiect generat" sau "obiect nou" şi semantica "obiect conservat" sau "obiect păstrat". Cu toate că în literatura de specialitate [Tresch 93], este negat faptul că o schemă externă poate să includă clase derivate prin semantica obiect generat, s-a dovedit că acest lucru este adevărat. S-a dezvoltat conceptul de clasă derivată pe baza conceptului fundamental "moştenire", prin aceasta înțelesul pe care îl are clasa derivată în cadrul unei baze de date orientate obiect sau relațional obiectuale capătă noi valențe

III. Pornind de la faptul că în [Catte 00] nu este prezentată o metodologie de definire a claselor derivate, s-a formulat un algoritm de definire a acestora. S-a schițat un algoritm de generare a schemelor externe în standardul ODMG. S-a încercat extinderea standardului [Catte 00], prin adaptarea lui la arhitectura ANSII/SPARC pe trei nivele. S-a propus realizarea unui limbaj specific pentru crearea schemei externe, prin aceasta s-a propus de fapt extinderea limbajului ODL al [Catte 97];

IV. S-a dat un sens mai larg noțiunii de "vedere", proprie bazelor de date relaționale, folosită în ORDBMS-uri. Ea devine clasă derivată şi este folosită la generarea schemei externe. Noțiunea de schemă externă temporală este o noțiune nouă şi a fost folosită în activitatea de proiectare a bazei de date;

V. Modul de abordare a mediului de memorare relațional. El devine obiectual, prin definirea clasei de obiecte care generează cod program prin care tabelele, relațiile dintre tabele şi vederile SQL, devin clase de obiecte;

VI. S-a realizat o clasificare nouă a Sistemul de Fabricație al Sculelor, pe baza căreia s-a realizat implementarea bazei de date. Pentru prima dată, pentru Sistemul de Fabricație al Sculelor, se face o analiză amănunțită a modului de

implementare, atât în mediul pur obiectual cât şi în mediul relațional obiectual. Implementarea în mediul relațional obiectual al Sistemul de Fabricație al Sculelor, definirea claselor de obiecte, a metodelor, a claselor derivate, folosirea schemelor externe în procesul de proiectare a bazei de date obiectuale, adică a conceptelor fundamentale analizate şi dezvoltate în teză, au fost implementate în baza de date realizată. Folosind avantajele bazelor de date orientate obiect, sau folosit obiecte multimedia necesare Sistemul de Fabricație al Sculelor. S-a realizat legătura Sistemul de Fabricație al Sculelor, cu medii CAD de proiectare asistată, folosind caracteristicile multimedia a bazelor de date orientate obiect.

Produsul soft realizat pe baza conceptelor dezvoltate în cadrul tezei poate fi folosit atât pe plan teoretic cât şi pe plan practic.

Din punct de vedere teoretic el poate fi folosit de către studenți în procesul de învățământ, atât la facultățile cu profil mecanic, pentru informațiile pe care le conține referitoare la Sistemul de Fabricație al Sculelor, cât şi la cele cu profil informatic pentru modul în care au fost tratate aspectele teoretice şi modul de implementare într-un mediu obiectual relațional al Sistemul de Fabricație al Sculelor.

Din punct de vedere practic, aplicația este folosită în cadrul secției sculărie din cadrul unei societăți comerciale cu profil mecanic. Ea oferă posibilități multiple atât din punct de vedere al consultării datelor alfanumerice şi cele multimedia, al execuției unor calcule necesare în activitatea de proiectare, în obținerea fişei tehnologice, în obținerea unor documente de natură economică sau în prelucrarea informațiilor obținute din medii CAD. Astfel din punct de vedere al utilității tezei în domeniul CAD, CAM se pot reliefa următoarele aspecte:

- Proiectarea asistată de calculator (CAD), include aplicații specifice diverselor domenii în care este folosită. În teză s-a pus accentul în special pe aspectul mecanic. Faptul că aplicația implementată, a permis reprezentarea directă a obiectelor complexe, a versiunilor de obiecte, a controlului direct asupra obiectelor face din sistemele de baze de date orientate obiect, atât cele hibride, cât şi cele pur obiectuale, instrumente ideale pentru această activitate;
- În domeniul producției asistată de către calculator (CAM), teza scoate în evidență rolul OODB-urilor în gestiunea producției. Ea scoate în evidență, gestionarea obiectuală a entităților, care apar în acest proces.

Pe scheletul tezei se derulează un contract de cercetare cu C.N.C.S.I.S. Totodată capitole din teză au făcut subiectul unor lucrări ştiințifice prezentate în țară la Timişoara şi Cluj precum şi în străinătate (Liubliana-Slovenia). Este în faza de definitivare, un contract de colaborare cu societățile Compa S.A. şi Indes din Sibiu, pentru implementarea aplicației la nivelul secției "Sculărie".

Perspective

Continuarea contractului cu Ministerul Educație şi Cercetării, va determina implementarea bazei de date obiectuale pentru sisteme de fabricație pe INTERNET. Din acest punct de vedere aspectul didactic va fi şi mai bine reliefat. Totodată vor creşte posibilitățile de informare, pentru cei care doresc implementarea unui astfel de software la nivelul unui atelier sau secție dintr-o societate economică

Aplicația implementată în societățile economice, va fi întreținută şi dezvoltată în continuare, prin aceasta se va căuta găsirea de noi soluții obiectuale, pentru rezolvarea problemelor legate de gestiunea, fabricarea şi proiectarea sculelor.

Obiectivele care vor fi urmărite, prin implementarea software-ului realizat, într-o societate economică din țara noastră sunt următoarele:

- simplificarea circuitului documentației;
- evitarea reproiectării unor scule;
- cunoaşterea exactă a stocurilor în orice moment;
- pregătirea mai rapidă a fabricației;
- reducerea stocului de scule (cu minim 20%);
- (respectarea) termenelor de livrare;
- calificarea mai înaltă a intreprinderii (cf. ISO 9000).

Soluțiile teoretice găsite privind folosirea schemelor obiectuale şi a claselor derivate, pentru sisteme de fabricație vor face subiectul unor lucrări ştiințifice, ce vor fi prezentate în anul 2003, la conferințe internaționale la Barcelona în Spania şi Podgorita în Muntenegru.

Cercetarea va continua în mediul distribuit, elaborându-se o carte despre bazele de date orientate obiect şi cele relațional obiectuale, cu particularizare spre sistemele de fabricație, carte ce lipseşte din literatura de specialitate din Romania.

Bibliografie

[Abite 88] Abiteboul, S., Hull, R., "Restructuring Hierarchical Database Objects", *Theoretical Computer Science*, 62,pp. 3-38, 1988;

[Abite 89] Abiteboul, S., Kanellakis, P., *Object identity as a query language primitive*, Proceedings of the 1989 ACM SIGMOD, Portland, Oregon, June 89;

[Abite 91] Abiteboul, s., Bonner, A., "Objects and Views", *Proc. ACM SIGMOD Int'l Conf. on Management of Data*,pp. 238-247, Denver, 1991;

[Abite 95] Abiteboul, S., Hull, R.,Vianu, V., *Foundations of Databases*, Addison-Wesley, 1995;

[Abrud 96] Abrudan, I., "Sisteme flexibile de fabricaţie. Concepte de proiectare şi management", Editura Dacia, Cluj, 1996;

[Alhaj 93] Alhajj, R., Arkun, M., "An Object Algebra for Object-Oriented Database Systems", *DATABASE*, vol. 24, no.3, pp. 13-22, August 1993;

[Amble 00-1] Ambler, S.W., *Mapping Objects To Relational Databases*, An AmbySoft Inc. White Paper 2000;

[Amble 00-2] Ambler, S.W., The Design of a Robust Persistence Layer For Relational Databases, An AmbySoftInc. White Paper 2000;

[Amble 01] Ambler, S.W. *The Object Primer 2nd Edition: The Application Developer's Guide to Object Orientation.* New York: Cambridge University Press, 2001;

[Amble 98] Ambler, S.W., *Building Object Applications That Work: Your Step-By-Step Handbook for Developing Robust Systems with Object Technology.* SIGS Books/Cambridge University Press, 1998;

[Ander 93] Andersen, J.,Reenskaug, T.,"Operations on Sets in anOODB", *OOPS Messenger*, vol. 4, no. 1, pp. 12-25January 1993;

[Andon 97] Andone, I., Tugui, A., "*Baze de date inteligente în managementul firmei*", Editura Dosoftei, Iasi 1997;

[ANSI 75] ANSI/X3/SPARC Study Group on Database Management Systems, "Interim report", *ACM SIGMOD*, bulletin 7, no. 2, 1975;

[ANSI 86] ANSI/X3/SPARC Database System Study Group, "Reference Model for DBMS Standardisation", *SIGMOD Record*, vol. 15, no. 1, pp. 19-58, March1986;

[Atkin 89] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrick, K., Maier, D., Zdonik, S., "*The Object Oriented Database System Manifesto*" 1989 ;

[Baner 87] Banerjee, J., Chou, H.T., Garza, J., W. Kim, D. Woelk, N.,Ballou, H.J. Kim, "*Data model issues for object-oriented applications*", ACM TOIS, January 1987;

[Baner 88] Banerjee, Won, Kim, K.,C., "*Queries in Object Oriented Databases*", Proc. 4th Intl. Conf. On Data Engineering, Los Angeles, CA, 1988;

[Barcl 93] Barclay, P., Kennedy, J., "Viewing Objects", *11th British National Conf. on Databases*, Springer, pp.93-109, Keele, July, 1993;

[Berti 92] Bertino, E., "A View Mechanism for Object-Oriented Databases", *Proc. Int'l Conf. on Extending Database Technology*, Springer, pp. 136-151, Vienna, March 1992;

[Blaha 95] Blaha, M., Premerlani, W., *"Object Oriented Modelling and Design for Database Aplications"*, Prentice Hall, Englewood Clifs, NJ, 1995;

[Booch 94] Booch, G., *"Object Oriented Analysis and design 2nd, Ed."*, Benjamin Cummings, Redwoord City, CA 1994;

[Borza 00-1] Borza, S., CA Jasmine *"Bazele de date intră în Mainstream"*, PC Report, Octombrie 2000;

[Borza 00-2] Borza, S., Inţă, M., *"Utilizarea calculatoarelor"*, Editura Universităţii "Lucian Blaga" din Sibiu, 2000;

[Borza 01-1] Borza, S., Brandaşu, D., P., *"Contribuţii privind implementarea unei baze de date orientate obiect pentru sistemul de gestiune al SDV-urilor, într-un mediu relaţional şi într-un mediu pur obiectual"*, Conferinţa ştiinţifică cu participare internaţională, Tehnomus XI, pag 85-90, Suceava, 2001;

[Borza 01-2] Borza, S., Brandaşu, D., P., *"Contribuţii privind implementarea obiectuală a unei baze de date pentru sistemul de gestiune al SDV-urilo"r*, Conferinţa ştiinţifică cu participare internaţională, Tehnomus XI, pag 91-96, Suceava Mai 2001;

[Borza 01-3] Borza, S., Brandaşu, D., P., *"The Object Implementation for The Administration System Cutting Tools"*, in Abstract Proceedings pag. 16, of Fifth International Conference on Management of Innovative Technologies, Fiesa-Piran Slovenia, MIT 2001;

[Borza 01-3] Borza, S., Brandaşu, P., D., *"The Contribution Looking of an Object Oriented Database for The Administration System Cutting Tools"*, in Annals of MteM 2001 & Proceedings of The 5th International MteM Symposium Cluj Napoca 2001;

[Borza 02] Borza, S., Brandaşu, P., D., *"The Aspects About The Administration of Tehnological Equipaments, in an Object System"*, in International Conference on Integrated Engineering, pag 53-54, Timişoara aprilie 2002;

[Borza 97] Borza S., Roşca N., Roşca L., *"Programarea în FoxPro. Teorie şi aplicaţii"*, Editura Universităţii "Lucian Blaga" din Sibiu 1997;

[Borza 97-2] Borza S., Roşca N., Roşca L., *"Baze de date. FoxPro"*, Editura Universităţii "Lucian Blaga" din Sibiu, 1998;

[Borza 98-1] Borza S., Cioca M., *"Sistemele informatice ale unităţilor economice"*, Editura Universităţii "Lucian Blaga" din Sibiu, 1998;

[Borza 99-1] Borza S., *"Proiectarea si Programarea bazelor de date Access"*, Editura Universităţii "Lucian Blaga" din Sibiu, 1999;

[Borza 99-2] Borza, S., Roşca, N., Roşca, L., *"Programarea în limbajul Turbo Pascal"*, Editura Universităţii "Lucian Blaga" din Sibiu, 1999;

[Borza 99-3] Borza, S., Roşca, N.," *Long Bones Fractures Management Training Database"*, AO International Documentation Center, Davos, Elvetia, 1999;

[Brats 92] Bratsberg, S., *"Unified Class Evolution by Object Oriented Views"*, Proc. Int'l Conf. on the Entity-Relationship Approach, pp. 423-439 , Karlsruhe,October 1992;

[Brèch 95] Brèche, P., Ferrandina, F., M. Kuklok, *"Simulation of Schema Change using Views"*, Proc. Int'l Conf. On Database and Expert Systems Applications, pp. 247-258, London, September 1995;

[Brind 01] Brindasu P., D., **Borza, S.**, *"Elaborarea unui sistem informatic pentru gestiunea sculelor aşchietoare, implementat pe o baza de date orientata obiect*, Analele Universitatii din Oradea, pag 24-31, 2001;

[Brind 94] Brindaşu, P.,D., *"Scule şi Portscule pentru Maşini Unelte cu Comandă numerică"*, Editura Universităţii "Lucian Blaga" Sibiu 1994;

[Buchh 84] Buchheit, M., Jeusfeld, M., Nutt, W., Staudt, M., "Subsumption between Queries to Object-Oriented Databases", *Information Systems*, vol. 19, no. 1, pp. 33-54, 1984;

[Caste 92] Castellanos, M., Saltor, F., García, M., "*A Canonical Model for the Interoperatibility among Object Oriented and Relational Models*", Proc. Int'l. Workshop on Distributed Object Management, pp. 309-314, Edmonton, Aug. 1992.

[Catte 00] Cattell, R.,G.,G., Barry, K., D., "*The Object Database Standard:Odmg 3.0*", Morgan Kaufmann Series 2000;

[Catte 97] Cattell, R.,G.,G., Barry, K., D., "*The Object Database Standard:Odmg 2.0*", Morgan Kaufmann Series 1997;

[Conno 01] Connoly, T., Begg, C., Strachan, A., "*Database Systems-A Practical Approach to design, Implementation and Management Secon Edition*", Addison Wesley Longman Limited 1995,1998, Editura Teora, 2001;

[Cook 94] Cook, S., Daniels, J., "*Designing Object Systems: Object Oriented Modelling with Syntropy*", Prentice Hall, Hemel, Hempstead, 1994;

[Darwe 98] Darwen H., Date C.J. "*Foundations for Object/Relational Databases: The Third Manifesto*", Harlow, Addison Wesley Longman, 1998;

[Date 00] Date, C., J., "*What do you mean, "post-relational"?*",2000, www.dbdebunk.com;

[Date 02] Date, C.,J., "*Models, Models, Everywhere, Nor Any Time To Think*", 2002, www.dbdebunk.com;

[Date 95] Date C.J., "*An Introduction to Database System*" Addison Wesley editia a 6-a 1995;

[Dayal 89] Dayal, U.,"*Queries and Views in an Object-Oriented Data Model*", Proc. 2nd Int'l. Workshop on Database Programming Languages, 1989;

[Dima 02] Dima, G., Dima, M., "*Visual FoxPro 7.0*", Editura Teora 2002;

[Ditri 91] Ditrich, K.,R., "*Object Oriented Database Systems: The Notion and the Issues*", Springer-Verlag Berlin Heidelberg 1991 pag 3-13;

[Dittr 88] Dittrich, K.R. (ed) "*Advances in Object-Oriented Database Systems*", Lecture Notes in Computer Science, Vol, 334, Springer-Verlag, 1988;

[Dittr 91] Dittrich, K.,R., Dayal, U., "*On Object oriented Database Systems*" Springer-Verlag, 1991;

[Dolli 98] Dollinger, R., "*Baze de date şi gestiunea tranzacţiilor*", Editura Albastra Cluj 1998;

[Du Boi 01] Du Bois, P., *MySQL*, New Readers Publishing, Editura Teora 2001;

[Fallo 99] Fallon, P., "*Building Jasmine Database An Introduction*", Jasmine Conference, CA-Word99 Session: JG105SA and JG105SB V1.00, July 1999;

[Fallo 99-2] Fallon, P., "*Why objects? And Why an Object Database?*", Jasmine Conference, CA-World99 session: JG106S V1.10, March 1999;

[Ferra 95] Ferrandina, F., Meyer, T., Zicari, R., Ferran, G., Madec, J., "*Schema and Database Evolution in the O2 Object Database System*", Proc. Int'l. Conf. on VeryLarge Databases, pp. 170-181, Zürich, Sep. 1995;

[Finga 98] Fingar, P., "*A CEO's Guide to eCommerce Using Intergalactic Object-Oriented Intelligent Agents*", Object News, London, 1998;

[Garda 96] Gardarin, G., Yoon, S., "*On the Power of Views in Hypermedia Databases*", *Proc. Engineering Systems Design and Analysis Conf.*, vol. 2, pp. 31-40, Montpellier, July 1996;

[Genti 94] Gentile, M., Zicari, R., " *Updating Views in Object Oriented Database Systems*", Proc. Int'l. Symposium on Adavanced Database Technologies and their Integration, Nara, Japan, October, 1994;

[Geppe 93] Geppert, A., Scherrer, S., Dittrich, K., R., "*Derived Types and Subschemas: Towards Better Support for Logical Data Independence in Object-Oriented Data Models*", Univ. Zürich, Institut für Informatik, Tech. Rep. 93.27, June, 1993;

[Goldb 83] Goldberg, A., and Robson, D., "*Smalltalk-80: the language and its implementation*", Addison-Wesley, 1983;

[Gottl 88] Gottlob, G., Paolini, P., Zicari, R., "*Properties and Update Semantics of Consistent Views*", ACM Transactions on Database Systems, vol. 13, no. 4, pp. 486-524, December 1988;

[Gray 92] Gray, P., Kulkarni, K., Paton, N., "*Object-Oriented Databases. A Semantic Data Model Approach*", Prentice Hall, 1992;

[Guerr 97] Guerrini, G., Bertino, E., Catania, B., García-Molina, J., "A Formal Model of Views for Object-Oriented Database Systems", , In Theory and Practice of Object Systems, Vol 3 pag 157-183, 1997;

[Gusta 94] Gustafsson, M., R., Karlsson, T., Bubenko, J., "*A Declarative Approach to Conceptual Information Modelling*", Information Systems Design Methodologies: A Comparative Review, pp. 93-143, 1994;

[Hamme 83] Hammer, H., McLeod, D., "*Database Description with SDM. A Semantic Database Model*", ACM Tranzaction on Database Systems 1983;

[Heile 88] Heiler, S., Zdonik, S., "*Views, Data Abstraction, and Inheritance in the FUGUE Data Model*", Proc. 2nd Int'l. Workshop on OODBS, Springer, FRG, Sep.,1988;

[Heuer 91-1] Heuer, A., Sander, P., "*Preserving and Generating Objects in the LIVING IN A LATTICE Rule Language*", Proc. Int'l IEEE Conf. on Data Engineering, pp. 562-569, Kobe, April 1991;

[Heuer 91-2] Heuer, A., Scholl, M., "*Principles of Object-Oriented Query Languages*", Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", pp. 178-197, Springer, Kaiserslautern, March 1991;

[Hoffe 02] Hoffer, Prescott & McFadden, "Modern Database Management", 6th Edition, 2002;

[Hogan 90] Hogan, R., "*A Practical Guide to Database Design*", Englewood Cliffs, Nj:Prentice Hall 1990;

[Honou 99] Honour, E., Dalberth, A., Kaplan, A., Mehta, A., "*Oracle8 How –To*", Editura Teora, 1999;

[House 02] Houser, J., "*Instant ColdFusion 5*", Osborn Media Group, Editura Teora 2002;

[Hull 84] Hull, R., Yap, C., "*The Format Model: A Theory of Database Organization*", *Journal of the ACM*, vol. 31, no. 3, pp. 518-537, July 1984;

[Hull 86] Hull, R., "*Relative Information Capacity of Simple Relational Database Schemata*", *SIAM Journal of Computing*, vol. 15, no. 3, pp. 856-886, August 1986.

[Hull 91] Hull, S. Widjojo, D. Wile, M. Yoshikawa, "*On Data Restructuring and Merging with Object Identity*", IEEE Data Engineering, vol. 14, no. 2, pp. 18-22, June 1991;

[ISO 92] ISO Database Language SQL (ISO 9075:1992(E)) International Organization for Standardisation 1992;

[ISO 98] ISO Database Language SQL Part 2 Foundation (ISO/IEC 9075-2) International Organization for Standardisation 1998;

[Jepso 00] Jepson, B., "*Object-Oriented Apps in a Relational Database*", New Architect Programming 2000;

[Jepso 02] Jepson, B., "*Object-Oriented Apps in a Relational Database*", CMP Media LLC, 2002;

[Jian 94] Jian, I., "*Utilizarea bazelor de date*", Imprimeria Mirton, Timişoara, 1994

[Johns 89] Johnson, D., Hyoung-Joo, K., *A minimal Object Oriented Data Model*. In Proceedings of the 27 th Annual Southeast Region ACM Conference, 1989;

[Kanda 01] Kanda, T., Mineo, K., *"New Trend of database for the Internet era – Object database and its application"*, ISES'01, martie 2001, Wuhn University;

[Katz 91] Katz, H., R., E-Li Chang, *Inheritence Issues in Computer Aided Design Databases*, Springer-Verlag Berlin Heidelberg 1991 pag 45-53;

[Kelle 98] Keller, W., Mitterbauer, C., Wagner K, "*Object-oriented Data Integration:Running Several Generations of Database Technology in Parallel*", in Akmal Chaudhri, Mary Loomis (Eds.), Object Databases in Practice, Prentice Hall 1998;

[Khosh 93] Khoshafian, S., "*Object Oriented Databases*", John Wiley&Sons, 1993;

[Kifer 92] Kifer, M., Kim, W., Sagiv, Y., *"Querying Object-Oriented Databases*" Proc. ACM SIGMOD Conference on Management of Data, pp. 393-402, San Diego, 1992;

[Kim 89] Kim, W.,"*A model of Queries for Object-Oriented Databases*", Proc. Int'l Conf. on Very Large Databases, pp. 423-432, Amsterdam, August 1989;

[Kim 95] Kim, W., Kelley, W.,"*On View Support in Object-Oriented Database Systems*", Modern Database Systems: the Object Model, Interoperability, and beyond, W. Kim (Ed.), pp. 108-129, ACM Press, 1995;

[Kimur 91] Kimura, Y.,Tsuruoka, K., "*A View Class Mechanism for Object-Oriented Database Systems*", Int'l Symp.on Database Systems for Advanced Applications, pp.269-273, Tokyo, April 1991;

[Kurim 88] Kurimo, A., So, K., Hill, D., "*Fully integraded tool management system for CIM*", Proccedings of 7-th International Conference on Flexibile manufacturing Systems", 1988;

[Leawi 00] Leawitt, N., "*Whatever Happened to Obect-Oriented Databases*", Lee Garber Inc, Los Alamitos 2000;

[Lemke 95] Lemke, T., *"DDL = DML ? An Exercise in Reflective Schema Management for Chimera"*,IDEA.WP.22.O.003, http://www.ecrc.de/IDEA/, March 1995;

[Marte 01] Martens, H., Rahm, E., "*On Parallel Join Processing in Object-Relational Database Systems*", University of Leipzig, White Paper 2000;

[Mengc 01] Mengchi, L., Shilpesh, K., "*Using Deductive Object-Relational Database in CAD*", DEXA 2001;

[Micro 99-1] Microsoft, Press, "*Microsoft VisualBasic 6.0 Programmer's Guide*" Editura Teora, Bucureşti 1999;

[Micro 99-2] Microsoft, Press, "*Microsoft VisualFoxPro 6.0 Programmer's Guide*" Editura Teora, Bucureşti 1999;

[Miller 94] Miller, R., Ioannidis, Y., Ramakrishnan, R., "*Schema equivalence in Heterogeneous Systems: Bridging Theory and Practice*", Information Systems, vol. 19, no. 1, pp. 3-11, 1994;

[Monk 94] Monk, S., "*View Definition in an Object-Oriented Database*", Information and Software Technology, vol. 36, no. 9, pp. 549-554, 1994;

[Monk 93] Monk, S., Sommerville, Y., "*Schema Evolution in OODBs Using Class Versioning*", SIGMOD Record, vol. 22, no. 3, pp. 16-22, September 1993;

[Motsc 96] Motschnig-Pitrik,R., "*Requirements and Comparison ofView Mechanisms for Object-Oriented Databases*", *Information Systems*, Elsevier, Vol. 21(3), 1996, pp. 229-252;

[Naja 95] Naja, H., Mouaddib, N., "*The Multiple Representation in an Architectural Application*", Proc. Int'l. Conf. On Database and Expert Systems Applications, pp. 237-246, London, September 1995;

[Olivé 89] Olivé, A.,"*On the Design and Implementation of Information Systems from Deductive Conceptual Models*", Proc. Int'l. Conf. Very Large Databases, pp. 3-11, Amsterdam, August 1989;

[OMG 97] OMG Common Object Request Broker Arhitecture and Specification, Object Management Group, Revision 2.1, 1997;

[OMG 92] OMG şi X/Open , "*CORBA Arhitecture and Specification*", Object Management Group, 1992;

[Peters 95] Peters, R., Özsu, M.,"*Axiomatization of Dynamic Schema Evolution in Objectbases*", *Proc. Int'l IEEE Conf. Data Engineering*, pp. 156-164, Tapei, March 1995;

[Popes 96] Popescu, I., "*Baze de date relaţionale*", Editura Universităţii din Bucureşti 1996;

[Popov 98] Popovici, M., D., Popovici, M., I., Rican, J., G., "*Proiectare şi implementare software*", Editura Teora, Bucureşti, 1998;

[Quer 94] Quer, C., Olivé, A., "Determining Object Interaction in Object-Oriented Deductive Conceptual Models", *Information Systems,* vol. 19, no. 3, pp. 211-227, 1994;

[Ra 95] Ra, Y., Rundensteiner, E., "*A Transparent Object-Oriented Schema Change Approach Using View Evolution*", *Proc. Int'l IEEE Conf. on Data Engineering*, pp. 165-172, Taipei, March 1995;

[Rahm 98] Rahm, E., "*Evaluation of object-relational database systems for fulltext retrieval*", University of Leipzig, White Paper 1998;

[Rahme 98] Rahmel D, "*SAMS Teach yourself Database Programming with Visual Basic 6 in 24 Hours*", SAMS Publishing, 1998;

[Rennh 97] Rennhackkamp, M., "*Extending Relational DBMS, DBMS* and *Internet Systems* "(http://www.dbmsmag.com/) Copyright © 1997 Miller Freeman;

[Rennh 97] Rennhackkamp, M., "*Extending Relational DBMSs*", Miller Freeman, Inc., Cape Town, 1997;

[Rob 95] Rob, P., Coronel, C., "*Database Systems Design, Implementation and Management*", Body &Fraser Publishing Company 1995;

[Roman 96] Roman., D. "*Ingineria programării obiectuale*", Editura Albastră, Cluj 1996;

[Runde 92-1] Rundensteiner, E., "*MultiView: A Methodology for Supporting Views in Object-Oriented Databases*", Univ. of Cal., Irvine, Tech. Rep. #92-07, Jan. 1992;

[Runde 92-2] Rundensteiner, E., "*A Class Integration Algorithm and its Application for Supporting Consistent Object Views*", Univ. of Cal., Irvine, Tech. Rep. #92-50, May 1992;

[Runde 92-3] Rundensteiner, E., "*MultiView: A Methodology for Supporting Views in Object-Oriented Databases*", Proc. Int'l Conf. on Very Large Databases, pp. 187-198, Vancouver, Aug. 1992;

[Runde 92-4] Rundensteiner, E., Bic, L., "*Automatic View Schema Generation in Object-Oriented Databases*", Univ. of Cal., Irvine, Tech. Rep. #92-15, Feb. 1992

[Samos 95] Samos, J., "*Definition of External Schemas in Object Oriented Databases*", Proc. Int'l Conf. on Object Oriented Information Systems, pp. 154-166, Dublin, Dec. 1995;

[Santo 94] Santos,C., Abiteboul, S., Delobel, C., "*Virtual Schemas and Bases*", Proc. Int'l Conf. on Extending Database Technology, pp. 81-94, Cambridge, March 1994.

[Santo 95] Santos, C., "*Design and Implementation of Object-Oriented Views*", Proc. Int'l Conf. on Database and Expert Systems Applications, pp. 91-102, London, Sep. 1995;

[Schew 92] Schewe, K., Schmidt, J., Wetzel, I., "*Identification, Genericity and Consistency in Object-Oriented Databases*", Int'l Conf. on the Entity-Relationship Approach, pp. 341-356, Karlsruhe, Oct. 1992;

[Schmo 83] Schmolze, J., Lipkis, T., "*Classification in the KL-ONE Knowledge Representation System*", The Eigth Int'l.Joint Conf. on Artificial Inteligence, vol. 1, pp. 330-332, Aug. 1983.

[Schol 91-1] Scholl, M.,H., Laasch, C., Rich, C., Schek, H., Tresch, M., "*The COCOON Object Model*", Univ. Ulm, Faculty of Computer Science, Rep. #193, Dec. 1992;

[Schol 91-2] Scholl, M.,H., Schek, H., "*Supporting Views in Object-Oriented Databases*", IEEE Data Engineering, vol. 14, no. 2, pp. 43-47, June 1991.

[Seth 98] Seth, G., *Modeling Object/Relational Databases*, Miller Freeman, Inc., Cape Town, 1998;

[Shaw 86] Shaw, G., Zdonik, S.",*A Query Algebra for Object-Oriented Databases*", Proc. Int'l IEEE Conf. on Data Engineering, pp. 154-162, Los Angeles, 1990.. Skarra, S. Zdonik, "*The Management of Changing Types in an Object-Oriented Database*", Proc. Int'l. Conf. on Object-Oriented Programming Systems and Languages Applications, pp. 383-495, Portland, Sep. 1986;

[Solom 99] Solomon, B., Baier, I., **Borza**, S., Brindasu, P., D., "*Computer aided fracture management training program*", al 4-Lea Congres EFFORT (Federația Europeană a Asociaților Naționale de Ortopedie - Traumatologie), Bruxelles, Belgia, iunie 1999.

[Staja 98] Stajano, F., "*A Gentle Introduction to Relational and Object Oriented Databases*", ORL Technical Report TR-98-2, 1998;

[Stöhr 00] Stöhr, T., Märtens, H., Rahm, E., "*Multi-Dimensional Database Allocation for Parallel Data Warehouses*", Proc. 26th VLDB Conf., Cairo, 2000;

[Stone 90] Stonebraker M., Rowe L., Lindsay B., Gray P., Carie Brodie M.L., Bernstein P. and Beech D., "*The Third generation database system manifesto (1990)*", Proc. ACM SIGMOD Conf;

[Stone 90] Stonebraker, M., "*Inclusion of New Types in Relational Database Systems*", In Proceedings of the second Data Engeneering Conference 1990.;

[Stone 96] Stonebraker M., "*Object-Relational DBMSs: The Next Great Wave*", San Francisco, CA: Morgan Kaufman Publishers Inc, 1996;

[Strebe 98] Strebe, M., Perkins, C, "*Internet Information Server 4 Study Guide*", Sybex Inc. 1998;

[Tanak 93] Tanaka, K., Yoshikawa, M., Ishihara, K., "*Schema Virtualization in Object-Oriented Databases*", Proc. Of the 4th Int'l. Conf. on Data Engineering, IEEE Computer Society Press, pp. 23-30, Feb., 1993;

[Tresc 91] Tresch, M., "*A Framework for Schema Evolution by Meta Object Manipulation*", Proc. Int'l Workshop on Foundations of Models and Languages for Data and Objects, pp. 1-13, Aigen, Sep. 1991;

[Tresc 92] Tresch, M., M. Scholl, "*Meta Object Management and its Application to Database Evolution*", Proc. Int'l. Conf. on Entity-Relationship Approach, pp. 299-321, Karlsruhe, Oct. 1992;

[Tresc 93] Tresch, M., M. Scholl, "*Schema Transformation without Database Reorganization*", *SIGMOD Record*, vol. 22, no. 1, pp. 21-27, March 1993;

[Wang 99] Wang, Q., Maier, D., Shapiro, L., "*Revisiting Reference Materialization Techniques for Object Query Processing*", Technical report CSE-99-004, Oregon Graduate Institute, 1999;

[Won 88] Won Kim, "*A foundation for object-oriented databases*", MCC Technical Report, 1988;

[Won 92] Won Kim, "*Object Oriented Database System: strengths and weaknesses*", Focus on OOBMS, 1992;

[Wynko 00] Wynkoop, S., "*Special Edition Using Microsoft SQL Server*", Que Corporation, Editura Teora 2000;

[Zdoni 90] Zdonik, Stanley, David, "*Readings in Object Oriented database Systems*", San Mateo, CA: Morgan Kaufmann, 1990;

[Zetu 98] Zetu, D., Carate, E., "*Sisteme flexibile de fabricaţie*", Editura Junimea, Iaşi, 1998.