PROBLEME ŞI SOLUŢII IN TEHNOLOGIA BAZELOR DE DATE ORIENTATE PE OBIECTE

ing. Radu Bercaru
ing.Şerban Voinea

Institutul de Cercetări în Informatică

REZUMAT

Apariţia unor aplicaţii noi, precum proiectarea şi ingineria asistate de calculator, ingineria software, sisteme de fabricaţie asistată, sisteme bazate pe cunoştinţe sau sisteme multimedia a scos în evidenţă limitările destul de serioase ale sistemelor tradiţionale de gestiune a bazelor de date. Cerinţele de a reduce costul de dezvoltare şi exploatare a acestor aplicaţii au determinat necesitatea unui salt calitativ fundamental în tehnologia bazelor de date. Suportul acestui progres îl constituie paradigma orientării pe obiecte, care a condus la apariţia şi dezvoltarea bazelor de date orientate pe obiecte. Lucrarea de faţa contureaza o imagine la nivel conceptual asupra abordării orientate pe obiecte şi analizează impactul acestor concepte asupra tehnologiei bazelor de date. Se face o prezentare generală a stadiului actual al cercetării în domeniul bazelor de date orientate pe obiecte şi a raportului acestora cu alte tipuri de baze de date. Sînt evidenţiate, de asemenea, cîteva direcţii majore de cercetare prin care se tinde către standardizarea unui model unic de date orientat pe obiecte.

Cuvinte cheie: Baze de date inteligente, programare orientată pe obiecte, date abstracte, obiecte compuse, mostenire ierarhică.

1. Introducere

În ultimele trei decenii, evoluţia tehnologiei bazelor de date a cunoscut patru generaţii de sisteme (sisteme de fisiere, sisteme ierarhice, sisteme de tip reţea, sisteme relationale).

Tranziţia de la o generaţie la alta a fost determinata de necesitatea de a ţine pasul cu costurile (in rapida creştere) de elaborare, întreţinere şi dezvoltare a programelor de aplicaţie. Ea s-a produs prin transferul unor funcţii repetitive, de rutina, din zona aplicaţiilor, în sarcina sistemului de gestiune a datelor.

Sistemele convenţionale (relaţionale şi prerelaţionale) au raspuns bine cerinţelor pentru care au fost proiectate şi anume, aplicaţiile de gestiune economică. Totuşi, din momentul în care tehnologia relaţională a ieşit din laborator şi a intrat pe piaţă s-au făcut simţite limitările ei destul de serioase. Cu alte cuvinte, au fost identificate o serie întreagă de aplicaţii dificil de implementat pe suportul sistemelor relaţionale: proiectare şi inginerie asistate de calculator, inginerie software, sisteme de fabricaţie asistată (CAD, CAE, CASE, CAM), sisteme bazate pe cunoştinţe (sisteme expert şi nuclee de sisteme expert), [31] sisteme multimedia care gestionează imagini, grafice, voce şi text, programe de analiză şi modelare ştiinţifică şi statistice etc.

Aceste aplicaţii prezintă serioase dificultăţi în ce priveşte modelarea datelor. Ele necesită în majoritatea cazurilor, facilităţi pentru modelarea şi gestiunea unor entităţi complexe imbricate (cum sînt obiectele utilizate în proiectare şi inginerie şi documentele compuse); un set mai bogat de tipuri de date, adică tipuri de date definite de utilizator şi date de dimensiuni mari, nestructurate (imagini, documente audio şi text); concepte semnatice frecvent utile (cum sînt relaţiile de generalizare şi agregare); conceptul de evoluţie în timp a datelor (dimensiunea temporală a datelor, versiunile) etc.

Aceste aplicaţii întîmpină şi alte dificultăţi fără legatură cu modelarea datelor. O parte dintre ele necesită operaţii de calcul intensive, pe un volum mare de date rezidente în memorie şi impun cerinţe de performanţă ce nu pot fi satisfăcute de sistemele de baze de date relaţionale şi prerelaţionale. Modul şi mediul de execuţie a unora dintre aplicaţii implică, de asemenea, tranzacţii de lungă durată, interactive şi cooperante.

Cerinţele de a reduce costul de dezvoltare şi exploatare a acestor aplicaţii au scos în evidenţă necesitatea unui salt calitativ, fundamental în tehnologia bazelor de date, adică a unei noi tranziţii de esenţă şi nu a unor extensii incrementale, ale capacităţilor sistemelor de baze de date existente. Suportul acestui progres în tehnologia bazelor de date îl constituie paradigma orientării pe obiecte, dezvoltată în domeniul limbajelor de programare orientate pe obiecte, o dată cu introducerea limbajului Smalltalk-80. Există două raţiuni majore, care susţin afirmaţia anterioară.

În primul rînd, paradigma orientării pe obiecte poate fi baza unui model de date, care să subsumeze modelele sistemelor de baze de date convenţionale. Un model de date orientat pe obiecte poate reprezenta nu numai date, relaţii şi restricţii asupra datelor, dar permite şi încapsularea datelor şi a programelor care operează asupra acestora şi oferă un suport unitar pentru tratarea tipurilor de date oarecare, definite de utilizator. Soluţiile pentru majoritatea problemelor legate de modelarea datelor în sistemele convenţionale sînt implicite în modelul orientat pe obiecte [2].

În al doilea rînd, paradigma orientării pe obiecte, prin noţiunile de încapsulare şi moştenire (reutilizare), este concepută să reducă dificultăţile de elaborare şi dezvoltare a sistemelor software complexe. Acesta este exact obiectivul care a determinat în ultimele trei decenii evoluţia tehnologiei gestiunii datelor, de la sistemele de fişiere la sistemele relaţionale.

Potenţialul sistemelor de baze de date orientate pe obiecte este, deci, clar. Totuşi, există încă în acest domeniu probleme serioase, care trebuie rezolvate. În primul rînd, tehnologia de dată recentă necesită o perioadă de maturizare. Majoritatea ofertelor comerciale existente suferă în măsură mai mică sau mai mare din punctul de vedere al funcţionalităţii şi/sau performanţei. O serie de firme au realizat şi lansat pe piaţă numai "maşini" de baze de date cu interfeţe proprii pentru programatorii de aplicaţii; instrumentele pentru dezvoltarea de aplicaţii abia

încep să apară. În al doilea rînd, modelarea obiectelor complexe și a relațiilor dintre ele, în aplicații din domeniile anterior menționate, induce în mod necesar un plus în complexitatea căreia trebuie să-i facă față utilizatorul. Aceasta implică dificultăți suplimentare în implementarea unor sisteme de baze de date orientate pe obiecte, de înaltă performanță.
În al treilea rînd, absența unui standard în domeniu împiedică adoptarea și răspîndirea rapidă a tehnologiei orientate în domeniu pe obiecte [18].

2. Paradigma orientării pe obiecte

Secțiunea de față vizează conceptele definitorii ale sistemelor orientate pe obiecte, concepte ce stau la baza dezvoltării sistemelor de gestiune a bazelor de date orientate pe obiecte. Aceste noțiuni fundamentează paradigma orientării pe obiecte, care exercită un foarte important impact asupra activității de programare în general.
Noțiuni fundamentale. Programarea orientată pe obiecte permite modelarea unei probleme în termeni asemănători modului de gîndire a omului. Ea a apărut ca o necesitate în rezolvarea unor aplicații noi, complexe, cum ar fi sistemele CASE sau CIM, aplicații pentru care programarea structurată nu mai face față în mod satisfăcător.
Componenta de bază într-o astfel de programare este obiectul. Un obiect poate reprezenta orice, de exemplu: un număr, un șir de caractere, un fișier, un editor de text, un program etc. Întocmai precum omul descompune o problemă complexă în subprobleme simple, tot așa obiectele se pot clasifica într-o ierarhie, pe baza unor proprietăți comune. Se creează, astfel, posibilitatea moștenirii unor proprietăți, fiecare obiect fiind caracterizat în mod explicit doar de proprietățile sale specifice.
Unui obiect i se atașează un set de operații corespunzătoare tipului său. De exemplu, obiectele reprezentînd numere au asociate operații aritmetice, obiectele reprezentînd șiruri de caractere au asociate operații simbolice (concatenări, extrageri de subșiruri etc.), obiectele reprezentînd structuri de date au asociate operații de memorare și regăsire a informației etc.
Obiectele comunică între ele prin mesaje. Mesajul reprezintă o cerere adresată unui obiect în scopul efectuării unei operații specifice lui. Mesajul specifică operația dorită, nu și modul de realizare a ei; acesta este determinat de obiectul căruia îi este destinat mesajul. Așadar, calculul reprezintă o caracteristică intrinsecă a obiectelor invocate prin mesaje.
Setul de mesaje la care un obiect poate să răspundă alcătuiește interfața obiectului cu restul obiectelor. Singurul mod de interacțiune al unui obiect este prin intermediul interfeței sale. Această proprietate permite ca implementarea unui obiect să nu depindă de reprezentarea internă a celorlalte obiecte, ci doar de mesajele la care el poate să răspundă. Mesajele asigură în același timp modularitatea unui sistem bazat pe programare orientată pe obiecte.
Spre deosebire de programarea structurată, care privește un program ca fiind alcătuit din structuri de date și algoritmi, programarea orientată pe obiecte privește o aplicație drept o mulțime de obiecte împreună cu mesajele de comunicare între ele. Acest stil de programare asigură o flexibilitate deosebită programelor, orice modificări ulterioare realizîndu-se printr-un efort minim.
Prin organizarea obiectelor în clase se încapsulează comportamentul obiectelor într-o manieră similară tipurilor de date abstracte. Totodată, prin ierarhizarea claselor, programarea orientată pe obiecte asigură posibilitatea unui acces partajat, precum și a unei reutilizări a codului existent. Codul are rolul de a stabili semantica structurilor de date; nemaifiind dispersat în cadrul programului, el poate fi refolosit în aplicații noi.
În ultimii cinci ani, o serie de limbaje orientate pe obiecte, ca de exemplu C++ [15], SMALLTALK-80 [14] sau OPAL (Gemstone) [11] au devenit comerciale. O serie de companii de software și-au modificat stilul de programare, orientîndu-l spre obiecte. Un exemplu notabil în acest sens este decizia de a utiliza C++ pe stațiile de lucru SUN AT & T. Alte companii, ca de exemplu MICROSOFT, au extins BASIC-ul cu facilități de programare orientate pe obiecte.
În domeniul modelelor de date și al limbajelor pentru baze de date, evoluția către paradigme mai expresive se manifestă în patru direcții mai importante:
a) **Reprezentarea spațiului obiectelor.** Modelele de date aflate la baza unor sisteme de tip IMS, CODASYL și SQL permit reprezentarea directă a diferitelor tipuri de obiecte. De exemplu, IMS reprezintă obiectele sub forma de arbori, CODASYL sub forma unor grafuri aciclice, iar SQL sub forma unor tabele normalizate. Forma generală de reprezentare a obiectelor este graful arbitrar, ce poate include și cicluri.
b) **Programarea declarativă.** Prin această modalitate, utilizatorul specifică informația dorită, nu și modul de obținere a ei. Un exemplu în acest sens îl constituie limbajul SQL.
c) **Abstractizarea și modularizarea.** SQL standard (ANSI), folosește noțiunea de modul pentru a genera entități independente de compilare și pentru a furniza o abstractizare care să permită separarea reprezentării interne de interfața cu lumea exterioară.
d) **Moștenirea.** Conceptul de moștenire introdus în modelele semantice de date, este folosit pentru organizarea spațiului obiectelor, pentru partajarea codului între modulele funcționale ale unui program și pentru extensia și reutilizarea codului existent.
Caracteristici de bază. Programarea orientată pe obiecte prezintă o serie de caracteristici, între care trei sînt esențiale [25]:

- tipuri de date abstracte;
- moștenire;
- identitatea obiectelor.

Să considerăm exemplul din fig.1, care reprezintă o

erarhie de mașini Ford, incluzînd modelul de bază și variante LX, BT și +.

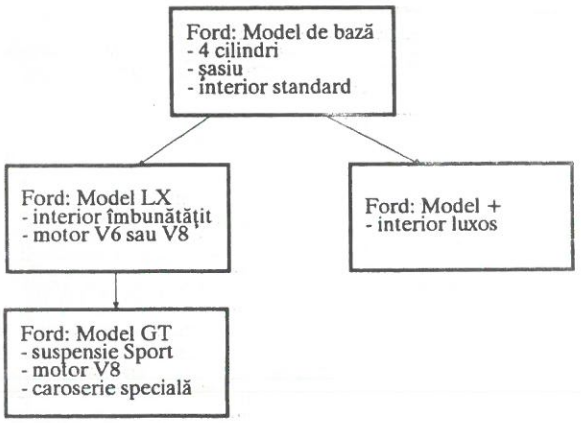

Fig.1

Orice mașină are o serie de trăsături comune. Schimbătorul de viteze împreună cu pedalele de acționare alcătuiesc interfața mașinii. Pentru a conduce o mașină, șoferul trebuie să acționeze numai pedalele și schimbătorul de viteze. El nu trebuie, de exemplu, să aibă cunoștințe despre structura interioară a motorului. Acest lucru evidențiază noțiunea de date abstracte, care separă o reprezentare internă de interfața cu alte obiecte.

Exemplul pune în evidență și ierarhia de mașini Ford. Astfel, modelul LX moștenește trăsăturile din modelul de bază, dar are ca elemente specifice un motor mai bun și un interior îmbunătățit. Aceasta este esența ierarhiilor, în care tipuri de obiecte moștenesc atribute de la obiectele generice, mai puțin specializate.

Fiecare mașină este alcătuită dintr-o serie de componente: motor, șasiu, sistem de suspensie, sistem de frînare, sistem de iluminare etc. Între aceste componente există diverse legături. De exemplu, un șasiu este partajat, atît de sistemul de suspensie frontal, cît și de cel dorsal. În acest mod, conceptul de mașină poate fi reprezentat sub forma unui graf, la ale cărui componente accesul și referirea se realizează prin intermediul identificatorilor.

Tipuri de date abstracte. Tipurile de date abstracte definesc similaritățile obiectelor împreună cu colecția de operatori asociați. Un limbaj care suportă tipuri de date abstracte trebuie să satisfacă următoarele cerințe:

a) **Definirea claselor.** Fiecare dată (obiect) trebuie să fie o instanță a unui tip abstract (a unei clase).
b) **Incapsularea informației.** Un obiect este accesibil și poate fi modificat doar prin intermediul interfeței sale (operatori definiți în tipurile de date abstracte). Detaliile de implementare sînt invizibile utilizatorului. Operațiile asociate cu tipurile de date abstracte asigură corectitudinea și completitudinea acestora.

În limbajele de programare convenționale, programele sînt văzute ca o colecție de proceduri ce comunică între ele prin intermediul parametrilor. În acest mod, procedurile sînt unitățile centrale de execuție.

În sistemele orientate pe obiecte, universul este văzut ca o colecție de obiecte care comunică între ele prin intermediul procedurilor (mesajelor). Așadar, în acest caz, obiectele sînt entitățile active, iar mesajele, împreună cu argumentele lor, sînt entitățile pasive, ce se transmit de la un obiect la altul. Un obiect răspunde mesajului recepționat. În consecință, datele reprezintă elementul central al programării orientate pe obiecte. Interfața reprezintă elementul public în cadrul programării orientate pe obiecte, în timp ce reprezentarea internă a obiectului rămîne un element privat. Orice actualizare a structurii unui obiect se poate realiza numai prin intermediul interfeței sale.

Clase. Clasa reprezintă modalitatea de definire a tipurilor de date abstracte prin limbajele de programare orientate pe obiecte. Ea încorporează definirea structurii interne a datelor, cît și a operațiilor permise asupra acestora. Colecția de obiecte descrise de o clasă poartă numele de instanțe ale clasei.

Definiția minimală a unei clase cuprinde următoarele:
a) numele clasei;
b) operațiile permise asupra instanțelor ei;
c) reprezentarea internă;
d) implementarea interfeței.

În fig.2 este prezentat un exemplu de definire a clasei Companie.

Operations:
GetProfit
GetSucursale
GetTotalAngajați
AddSucursala

Instance Variables
Nume
Adresa
Cheltuieli
Angajați
Sucursale

Fig.2

Starea internă a unui obiect este descrisă prin valorile variabilelor de instanță. Ele sînt specifice fiecărui obiect și, deci, fiecare instanță a unei clase va avea alocat un spațiu de memorie corespunzător stării sale. În schimb, toate instanțele unei clase partajează codul ce implementează operatorii clasei. Așadar, va exista un singur cod pentru implementarea operatorilor GetProfit, GetSucursale, GetTotalAngajați și AddSucursala. Toți acești operatori sînt invocați ca avînd drept obiect destinație o instanță a clasei Companie. Acest lucru este similar apelurilor de proceduri din limbajele de programare convenționale.

Variabile de instanță. Variabilele de instanță captează structura internă a unui obiect. În general, valorile acestor variabile sînt restricționate ca aparținînd unei anumite clase. În acest mod, pot fi detectate anumite

tipuri de erori încă din faza compilării. Sînt, în schimb, unele limbaje, ca de exemplu SMALLTALK, LISP sau APL, care nu impun această restricție. Astfel, o variabilă de instanță poate memora la un moment dat o valoare întreagă, dar aceeași variabilă de instanță poate memora ulterior o valoare de tip șir de caractere. Astfel de limbaje amînă detectarea eventualelor erori pînă în momentul execuției.

Metode și mesaje. Invocarea unui operator implică următoarele:

a) obiectul destinatar;
b) numele operatorului;
c) argumentele operatorului;
d) legarea argumentelor cu valorile parametrilor actuali și invocarea codului care implementează operatorul.

Un operator este implementat prin intermediul unei metode care corespunde definiției de procedură din limbajele de programare convenționale. Invocarea unui operator se realizează prin intermediul unui mesaj care este corespondentul apelului de procedură. Principala diferență dintre un apel de procedură și un mesaj constă în faptul că, în acest din urmă caz, legarea parametrilor se face în momentul execuției.

Legarea dinamică. Legarea dinamică constă în alegerea metodei ce implementează un operator la momentul execuției, în funcție de obiectul destinatar al mesajului. Acest lucru este necesar deoarece:

a) metode cu același nume pot exista în clase diferite;
b) în limbajele fără tipuri de date, tipul asociat unei variabile este cunoscut numai la momentul execuției.

Să considerăm că avem într-o stivă o serie de obiecte de tipuri diferite pe care dorim să le afișăm și ca fiecare tip de obiect are asociată o metodă Print specifică. În acest caz, instrucțiunea care realizează acest lucru se scrie foarte simplu, după cum urmează:

for i := 1 to TopOfStack do Print (Stack[i]).

În acest fel, fiecare obiect Stack[i] va apela metoda de afișare Print, specifică lui. Deci, metoda Print nu identifică un cod unic. Mesajul Print, adresat unui anumit tip de obiect, identifică însă un cod unic.

Prin legarea dinamică, sistemul poate fi însă extins prin simpla adăugare de clase noi și metode, fără a le afecta pe cele existente. Așadar, extensibilitatea și flexibilitatea sînt două mari avantaje ale acestei strategii.

Moștenire. Moștenirea reprezintă al doilea concept important, ce caracterizează un sistem orientat pe obiecte. Prin moștenire se pot construi clase noi, pornind de la cele vechi, care sînt mai puțin specializate. Clasele noi moștenesc, atît operațiile, cît și structurile claselor existente.

De exemplu, o companie comercială și o organizație fără profit au în comun o serie de elemente cum ar fi un nume, o adresă, personalul și așa mai departe. Organizațiile fără profit conțin o serie de informații specializate, ca de exemplu alocațiile guvernamentale sau legăturile internaționale cu organizații similare. În același mod, companiile comerciale pot avea informații specializate, ca de exemplu cifra de afaceri.

Pentru a îngloba informațiile comune vom construi, în consecință, o clasă generică numită Companie, avînd drept subclase Companie Comercială și OrganizațieNonProfit.

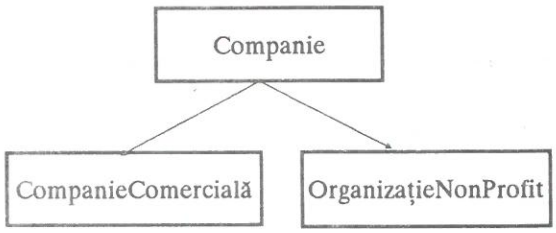

Fig.3

Orice instanță a unei clase este implicit și instanță a superclasei sale. Reciproca nu este însă adevărată. De exemplu, pot exista instanțe ale clasei Companie, care să nu fie nici instanțe ale subclasei CompanieComercială și nici ale subclasei OrganizațieNonProfit.

Relația de ierarhizare este tranzitivă. De exemplu, dacă SocietateSemiconductoare este o subclasă a lui CompanieComercială, atunci, prin tranzitivitate, ea este și subclasă a lui Companie. În consecință, SocietateSemiConductoare va moșteni structura internă și operațiile ambelor clase.

Moștenirea variabilelor de instanță. Variabilele de instanță pot fi accesate în două moduri:

a) numai prin metodele proprii ale clasei în care s-au definit variabilele de instanță;
b) prin metodele clasei în care s-au definit variabilele de instanță, precum și ale tuturor subclaselor acesteia.

Metoda a doua (specifică și pentru SMALLTALK) are dezavantajul că violează principiul uniformității și al încapsulării. În cazul în care se modifică implementarea unei variabile de instanță (de exemplu, se trece de la o structură de tip matrice la una de tip listă), atunci vor trebui modificate corespunzător și toate metodele de acces la această variabilă din toate subclasele.

Metoda generală, care combină eficiența cu flexibilitatea constă în stabilirea de atribute care determină modul de acces al variabilelor de instanță, după cum urmează:

a) public (variabila poate fi accesată de oriunde);
b) privat (variabila poate fi accesată numai din clasa în care a fost definită);
c) protejat (variabila este accesibilă din clasa în care a fost definită și din subclasele acesteia).

Cu aceste trei opțiuni, utilizatorul poate fi de la început restrictiv și poate impune ca toate variabilele și metodele să fie private. Pe măsură însă ce codul său devine stabil, metodele și variabilele de instanță pot fi făcute vizibile (publice sau protejate, cîștigîndu-se astfel în eficiență.

Moștenirea metodelor. Metodele se pot moșteni în același sens ca și variabilele de instanță [20]. În plus, o metodă dintr-o clasă poate fi acoperită de către o

metodă cu același nume, dintr-o subclasă a ei.
Uneori se dorește ca efectul suprapunerii metodelor să fie anulat. De aceea, este preferabil ca, o dată cu invocarea unei metode, să se precizeze și numele clasei din care face ea parte. Să considerăm, de exemplu, că în ierarhia de clase de companii avem și clasa CompanieComercialăStrăină, ca subclasa a lui Companie Comercială. Evaluarea profitului unei astfel de companii se realizează în mod diferit față de cazul unei companii autohtone. Drept urmare, în cele două clase vom avea două metode cu același nume pentru evaluarea profitului, cea din clasa CompanieComercialăStrăină acoperind-o pe cea din clasa CompanieComercială.
Moștenirea multiplă. Sînt situații cînd este convenabil ca o clasă să poată moșteni simultan de la mai multe clase. De exemplu, un student seralist va moșteni, atît caracteristicile unui angajat, cît și ale unui unui student (fig.4).

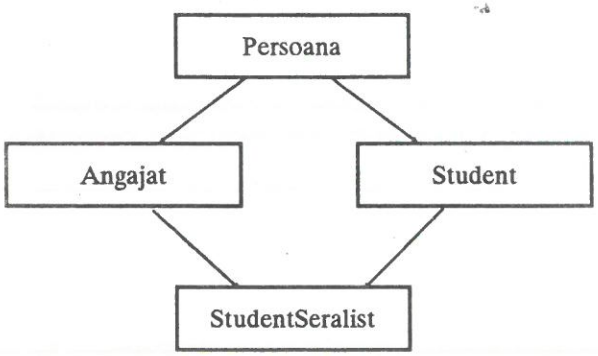

Fig.4

Prin această modalitate de moștenire multiplă se pot genera noi clase din cele existente, fiecare din ele utilizînd superclasele în funcție de necesitățile proprii. O instanță a clasei StudentSeralist va avea starea și comportamentul atît ale clasei Angajat, cît și ale clasei Student. De regulă, într-o moștenire multiplă, variabilele de instanță și metodele unei subclase sînt un superset ale reuniunii variabilelor de instanță și metodele superclaselor. De exemplu, dacă variabilele de instanță ale clasei Angajat sînt:

{nume, vîrstă, salariu, departament},

iar cele ale clasei Student sînt

{nume, vîrstă, cursuri, preparator},

atunci clasa StudentSeralist va avea următoarele variabile de instanță:

{nume, vîrstă, salariu, departament, cursuri, preparator}.

În cadrul unei moșteniri multiple pot apărea situații de conflict. Prin conflict înțelegem că, variabile de instanță sau metode cu același nume pot fi moștenite de la superclase paralele. Să presupunem, de exemplu, că, atît clasa Student, cît și clasa Angajat au fiecare definite cîte o metodă Supliment cu semantici total diferite. Care din aceste două metode va fi moștenită de clasa StudentSeralist ?
Există două tehnici de rezolvare a acestor conflicte:

a) prin redefinire;
b) printr-o ordine, implicită sau impusă, de moștenire a superclaselor.

Prin exemplul considerat, impunînd ca ordine implicită ordinea de definire a claselor, clasa Student seralist va moșteni metoda Supliment de la clasa Angajat.

Identitatea obiectelor. Identitatea constituie un mijloc de a distinge un obiect de altul. Într-un sistem orientat pe obiecte, fiecare obiect are asociată o identitate, indiferent de modificările structurale ale acestuia. Fără o identitate nu este posibilă stabilirea unui obiect drept componentă a altui obiect [16].
O modalitate de identificare a unui obiect, folosită des în limbajele de programare, o consituie atribuirea de nume. Totuși, această modalitate are unele limitări în sensul ca un obiect poate fi accesat în mai multe moduri, și, deci, pot exista mai multe variabile care să refere, de fapt, un același obiect.
Sistemele de baze de date folosesc pentru identificare chei ale înregistrărilor. În modelul relațional, cheia reprezintă chiar un atribut din înregistrare (de exemplu numele și prenumele unei persoane, modelul unei mașini, contul unei bănci etc.) Există anumite inconveniente în acest sens, și anume:

a) modificarea cheilor: valoarea unor cîmpuri dintr-o înregistrare se poate modifica de-a lungul timpului (de exemplu, o persoană poate să-și schimbe numele);
b) eterogenitatea: două firme își pot identifica angajații prin numere ce au, însă, semnificații diferite. O unire a celor două firme va crea dificultăți în identificarea angajaților.
c) joncțiuni nenaturale. Să presupunem că avem relațiile:

Angajat (NumeAng, Salariu, NumeComp)

Companie (Nume, Buget, Adresa)

Atributul NumeComp stabilește legătura dintre cele două relații (dintre un angajat și o companie). Utilizarea cheilor pentru identificare face ca o regăsire pe ambele tuple să necesite joncțiunea lor, în locul unei expresii mai simple. De exemplu, în SQL, pentru a regăsi numele angajaților și locul unde lucrează, va trebui folosită următoarea secvență:

```
SELECT Angajat.NumeAng, Companie.Adresa
FROM Angajat, Companie
WHERE Angajat.NumeComp = Companie.Nume
```

Joncțiunea este nenaturală, deoarece în majoritatea cazurilor utilizatorul este interesat să obțină tuplul asociat companiei respective, și nu numele ei.
Un aspect important, în cazul bazelor de date, constă în partajarea obiectelor. Partajarea implică sincronizarea acceselor concurente pentru asigurarea consistenței informației din baza de date. O bază de

date este accesată și actualizată prin intermediul trazacțiilor. Serializarea tranzacțiilor este realizată prin mecanismul de zăvorîre. Într-un sistem orientat pe obiecte, utilizatorul devine el însuși un obiect cu referințe către entitatea partajată. Un obiect ce devine inaccesibil pentru orice alt obiect va fi colectat în spațiul disponibil. Așadar, identitatea obiectelor este și un mijloc pentru referirea obiectelor partajate.
Avînd, astfel, conturată o imagine de nivel conceptual asupra abordării orientate pe obiecte din perspectiva limbajelor de programare, în cele ce urmează vom analiza impactul acestor concepte asupra tehnologiei bazelor de date.

3. Baze de date orientate pe obiecte

Un sistem de baze de date orientat pe obiecte este un sistem de baze de date ce suportă un model de date orientat pe obiecte. Ca orice sistem de baze de date, el trebuie să asigure persistența datelor în memoria secundară și să aibă integrat un limbaj pentru definirea și modificarea schemei, precum și o interfață care să asigure crearea și accesarea obiectelor.
Sistemul de bază trebuie să poată fi extins cu noi capacități, întocmai ca si sistemele relaționale. În primul rînd, un limbaj de cereri trebuie să poată fi definit în cadrul modelului de date orientat pe obiecte. În al doilea rînd, trebuie să existe posibilitatea adăugării de facilități care să asigure integritatea datelor în timpul tranzacțiilor. În al treilea rînd, sistemul trebuie să poată fi extins cu elemente care să asigure creșterea performanțelor sale, ca de exemplu, indexarea și gruparea (clustering) datelor în memoria secundară. În al patrulea rînd, sistemul trebuie să poată asigura controlul concurent la resursele sale în cadrul unui mediu de programare multiutilizator. În sfîrșit, sistemul de bază trebuie să poată fi extins cu concepte de modelare a datelor pentru crearea versiunilor de obiecte, precum și a obiectelor compuse.
Conceptele de bază ale orientării pe obiecte au fost prezentate în secțiunea anterioară. În cele ce urmează, vom explica impactul acestor concepte asupra bazelor de date.
Tratarea uniformă a entităților lumii reale drept obiecte simplifică în mod considerabil viziunea utilizatorului asupra lumii reale. Obiectele sînt referite și regăsite prin intermediul identificatorilor lor. Identificatorii au fost introduși în sistemele orientate pe obiecte din două considerente. Primul constă în faptul că, valorile atributelor unui obiect sînt, la rîndul lor, obiecte. În acest mod, reprezentarea naturală a stării unui obiect constă dintr-o mulțime de identificatori corespunzători valorilor atributelor sale. Din rațiuni de performanță, în cazul în care domeniul unui atribut este o clasă primitivă (un număr, un șir, etc), valoarea sa va fi reprezentată în mod direct, și nu prin intermediul unui identificator. De exemplu, domeniul atributului "Adresa" al clasei "Companie" din fig. 2 este clasa "Șir de caractere" și, ca atare, valoarea sa poate fi șirul

"Bd. Mareșal Averescu, 8-10, București, Rom. 71316"

Al doilea considerent se referă la faptul că, fundamentele orientării pe obiecte au fost concepute independent de cele ale bazelor de date. Ca atare, obiectele au fost presupuse ca aparținînd unei memorii virtuale, și, deci, identificatorul este atunci singurul mijloc de specificare al lor. Noțiunea de cerere pentru selectarea unui set de obiecte care satisfac o anumită condiție nu a fost în atenția proiectanților de limbaje orientate pe obiecte. Acest lucru a condus la stabilirea unui model procedural de prelucrare în majoritatea aplicațiilor orientate pe obiecte. Nu înseamnă însă că sistemele orientate pe obiecte nu pot fi înzestrate și cu modele declarative de prelucrare a obiectelor. Exemple în acest sens sînt sistemele ORION [7], 02 [32], GemStone [11] și IRIS [34]. De altfel, se pare că aceasta este o caracteristică a sistemelor viitoare, orientate pe obiecte.
Atributul unui obiect este corespondentul unui cîmp al unei relații din bazele de date relaționale. Domeniul unui atribut poate fi orice clasă definită de sistem sau de utilizator. Acesta reprezintă o diferență esențială față de modelul relațional normalizat, unde domeniul unui atribut este restrîns la o clasă primitivă. În acest mod, definiția unei clase se reduce la un graf de clase, care are ca rădăcină acea clasă. Dacă graful este strict ierarhic, atunci el are drept corespondent o relație imbricată. Acest graf poartă numele de ierarhie compozițională de clase (class-composition hierarchy) și el este ortogonal conceptului de ierarhie de clase, care captează relațiile de generalizare și specializare. În mod evident, în cadrul unei ierarhii compoziționale de clase pot exista legături ciclice. În fig.5 este prezentat un astfel de exemplu (liniile continue reprezintă ierarhia de clasă, iar cele intrerupte ierarhia compozițională de clase).

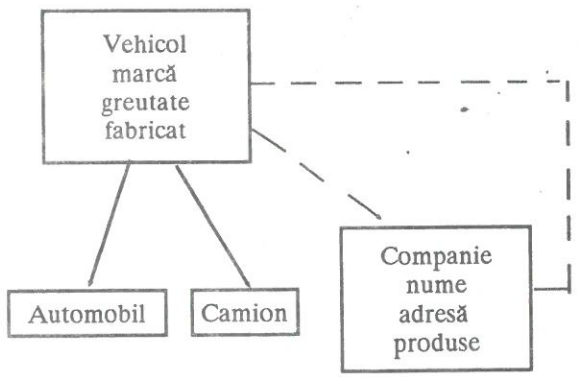

Fig.5

Conceptul de clasă consitituie legătura cea mai puternică dintre sistemele orientate pe obiecte și bazele

de date. În primul rînd, prin clasă se captează aspectele semantice ale datelor [6]. În al doilea rînd, clasa este la baza formulării cererilor: o cerere este emisă asupra unei clase sau a unui set de clase (excepție face GemStone care emite cereri asupra instanțelor unei clase). În al treilea rînd, conceptul de clasă asigură integritatea datelor, specificarea domeniului unui atribut drept o clasă, restricționează valorile atributului respectiv la obiecte care aparțin acelei clase. În al patrulea rînd, se realizează o evidentă economie de memorie pentru reprezentarea atributelor și metodelor obiectelor aparținînd unei aceleiași clase (în absența claselor, toate specificațiile referitoare la integritatea datelor ar trebui multiplicate pentru fiecare obiect).
Interfața unui sistem de baze de date orientat pe obiecte cu utilizatorul poate fi realizată în două modalități. Prima modalitate constă în definirea unui limbaj de baze de date, care să fie înglobat în limbajul sistemului (este și cazul sistemelor relaționale). Principalul dezavantaj constă în faptul că utilizatorul este obligat să învețe două limbaje diferite. A doua metodă constă în extensia limbajelor de programare, orientate pe obiecte, cu facilități specifice bazelor de date. Această metodă are avantajul că utilizatorul este nevoit să învețe numai niște extensii ale limbajului inițial, și nu un limbaj cu totul nou. Astfel de sisteme sînt ORION (ca extensie a lui Common LISP) [17], ZEITGEIST (ca extensie a lui Flavors), GemStone și AllTalk [23] (cu extensii ale lui SmallTalk). Operațiile de bază pentru bazele de date vor trebui, printre altele, să aibă în vedere și posibilități pentru definirea de clase, pentru selectarea, actualizarea sau ștergerea unei clase avînd anumite proprietăți, pentru transmiterea de mesaje.

4. Raportul bazelor de date orientate pe obiecte cu alte tipuri de baze de date

Limitările actuale ale bazelor de date convenționale, au dat un imbold important dezvoltării cercetărilor în domeniul proiectării unei noi generații de baze de date [10] capabile să prelucreze obiecte complexe, cu semantică bogată. Acest lucru s-a dovedit necesar în special în sistemele CAD/CAE, unde conceptele programării orientate pe obiecte trebuie integrate în cadrul unei baze de date, necesară prelucrării unui volum mare de obiecte complexe. În cele ce urmează, vom face o scurtă paralelă între bazele de date orientate pe obiecte și celelalte tipuri de baze de date.

Baze de date ierarhice și de tip rețea. O similitudine între cele două tipuri de baze de date constă în structura imbricată a obiectelor, respectiv a înregistrărilor. De asemenea, ambele tipuri de baze de date admit obiecte, respectiv înregistrări, care referă alte obiecte, respectiv înregistrări, drept valori pentru atributele lor. Diferența constă în faptul că, în timp ce bazele de date orientate pe obiecte admit cicluri [7] în mod natural, cele ierarhice necesită introducerea în acest scop a unor înregistrări artificiale (virtuale).
O altă similitudine constă în referirea la identificatorii obiectelor din bazele de date orientate pe obiecte și pointerii din înregistrările bazelor de date ierarhice. Identificatorul unui obiect este un pointer logic, și nu este refolosit niciodată; acest lucru asigură integritatea datelor. În schimb, pointerii de înregistrări sînt pointeri fizici și ei pot fi refolosiți.
Diferența esențială dintre cele două tipuri de baze de date constă în faptul că bazele de date ierarhice nu includ concepte evoluate precum clasă, moștenire sau metodă.

Baze de date extensibile. Elementul comun cu bazele de date orientate pe obiecte, îl constituie extensibilitatea. Problema esențială a bazelor de date extensibile este înzestrarea lor cu facilități de adăugare de noi funcționalități sau de construire a unei baze de date din componente prefabricate, memorate într-o bibliotecă corespunzătoare. Dacă un sistem de baze de date este implementat într-un stil orientat pe obiecte, atunci extensibilitatea se obține foarte simplu, pe baza posibilităților de moștenire prezente. De altfel, extensibilitatea este, mai degrabă, o caracteristică generală a arhitecturii unui sistem de baze de date, decît o cerință a unui sistem de baze de date orientat pe obiecte.

Baze de date semantice. Modelele de date semantice (modelul entitate-relație, modelul funcțional DAPLEX) încearcă reprezentarea complexității relațiilor semantice dintre entitățile lumii reale. Relația de generalizare/specializare dintre o clasă și subclasele ei, relația de agregare dintre o clasă și atributele ei, relația de instanțiere dintre o clasă și instantele ei sînt toate incluse în modelele de date semantice.
Modelul nucleului unui sistem de baze de date orientat pe obiecte, poate fi văzut ca un subset al unui model de date semantic; în mod evident, modelul de date semantic nu este înzestrat cu metode. Din considerente de performanță, modelul unui sistem de baze de date orientat pe obiecte, trebuie extins cu concepte de modelare semantică, dintre care cele mai importante sînt cele referitoare la versiuni și la obiecte compuse. Aceste concepte permit utilizatorului să privească o colecție de obiecte, legate între ele prin relații, drept o singură entitate. De exemplu, un obiect complex este o colecție de obiecte legate prin relația "compus-din" și el poate fi folosit drept unitate de acces în cadrul unei cereri sau ca unitate de integritate.

Baze de date relaționale. Între cele două modele de baze de date există diferențe clare [8]. Conceptele de ierarhie de clase, ierarhie compozițională de clase și metode nu pot fi incluse în modelul relațional normalizat. Se fac eforturi pentru extensia modelului relațional cu astfel de concepte, cel mai bun exemplu în acest sens constituindu-l sistemul POSTGRES [27,30].

Ceea ce se reproşează, în general, bazelor de date orientate pe obiecte este modelul lor procedural şi faptul că nu au la bază o teorie matematică elegantă precum cea care fundamentează modelul relaţional. Cu toate acestea, bazele de date orientate pe obiecte continuă să constituie un domeniu interesant de cercetare, existînd o serie de încercări pentru definirea unui model de cereri şi a unei algebre orientate pe obiecte (echivalentul algebrei relaţionale) [9].

5. Stadiul actual al cercetării în domeniul bazelor de date orientate pe obiecte

Există două tendinţe, în ceea ce priveşte dezvoltarea bazelor de date orientate pe obiecte, şi anume:

a) integrarea unui limbaj de programare orientat pe obiecte într-un sistem de baze de date;
b) proiectarea de arhitecturi de sisteme de baze de date.

Prima abordare presupune, de fapt, extensia unui limbaj de programare cu facilităţi specifice bazelor de date. Astfel, GemStone [22] şi AllTalk au extins limbajul SmallTalk în acest sens, în timp ce ORION şi Statice [33] funrizează un suport de baze de date pentru extensiile orientate pe obiecte ale limbajului Common LISP. Vbase [1], la rîndul său, oferă un suport de baze de date pentru limbajele de programare orientate pe obiecte COPS şi TDL. Tehnica de bază, utilizată pentru a suporta programarea persistentă în aceste sisteme constă în gestiunea spaţiului de lucru. În acest sens, există un director cu obiectele aflate în memoria de lucru, fiecare obiect avînd un descriptor corespunzător. Un obiect este referit de către celelalte obiecte prin intermediul descriptorului său. Descriptorul unui obiect conţine un pointer către obiectul respectiv, împreună cu alte informaţii de control, menite să îmbunătăţească performanţele sistemului.

Cea de a doua abordare presupune ca proiectanţii sistemelor de baze de date orientate pe obiecte au în vedere realizarea unor arhitecturi corespunzătoare, inluzînd tehnici de memorare a obiectelor, de control al concurenţei, de recuperare din eroare, de prelucrare a cererilor, de definire şi modificare a bazei de date, de autorizare a accesului. ORION, GemStone şi IRIS sînt sistemele care furnizează cele mai multe facilităţi în acest sens. De altfel, majoritatea tehnicilor folosite în implementarea sistemelor convenţionale de baze de date sînt aplicabile şi sistemelor de baze de date orientate pe obiecte. Totuşi conceptele programării orientate pe obiecte necesită unele extensii ale tehnicilor convenţionale. În cele ce urmează, ne vom concentra atenţia asupra impactului programării orientate pe obiecte, asupra arhitecturilor sistemelor de baze de date.

În sistemele orientate pe obiecte, fiecare obiect are asociat un identificator unic. Un astfel de identificator cuprinde, de fapt, o pereche:

<identificator clasă, identificator instanţă>

unde identificatorul de clasă reprezintă clasa de care aparţine obiectul, iar identificatorul de instanţă reprezintă numărul instanţei în cadrul clasei sau al întregii baze de date. Prin intermediul identificatorului de clasă, sistemul determină validitatea unui mesaj transmis şi, în caz afirmativ, asociază obiectului metoda adecvată. Din considerente de performanţă, aceste verificări trebuie făcute în timpul compilării. Identificatorul de clasă restricţionează, totuşi, mobilitatea unui obiect atunci cînd el trebuie să migreze de la o clasă la alta. În acest caz, o dată cu modificarea identificatorului de clasă, trebuie actualizate şi toate referinţele către acel obiect, ceea ce este un proces destul de costisitor.

Modelul unei baze de date orientată pe obiecte are două dimensiuni. Prima este ierarhia de clase care captează relaţia de generalizare dintre o clasă şi subclasele ei. Cealaltă dimensiune este ierarhia compoziţională de clase, care reprezintă relaţia de agregare dintre o clasă şi domeniile atributelor ei. Altfel spus, orice clasă într-o bază de date orientată pe obiecte aparţine simultan unei ierarhii de clase şi unei ierarhii compoziţionale de clase. Semantica ierarhiei de clase este cea care complică actualizările din model. De exemplu, dacă o clasă este ştearsă din ierarhie, toate subclasele ei pierd atributele şi metodele moştenite de la acea clasă şi, implicit, instanţele subclaselor îşi pierd valorile atributelor. În mod analog, adăugarea unei clase în ierarhie implică moştenirea atributelor şi metodelor superclaselor ei. Modificările în ierarhia de clase pot fi mascate în cadrul instanţelor (cazul sistemului ORION) sau, dimpotrivă, să fie reflectate imediat, ca în cazul sistemului GemStone.

Un dezavantaj al modelului bazelor de date orientate pe obiecte îl constituie faptul că presupune existenţa unei singure scheme. În acest mod, toate modificările făcute de utilizator asupra schemei vor fi automat văzute şi de ceilalţi utilizatori. Soluţia în acest sens constă, fie în accesul privat al administratorului bazei la modificarea schemei, fie în păstrarea mai multor versiuni ale schemei (scheme virtuale), astfel încît utilizatori diferiţi să vadă în mod diferit baza de date. Această din urmă soluţie a fost propusă iniţial în proiectul Observer [29] şi aplicată şi în ORION.

Modelul de cereri dezvoltat în ORION este unul din puţinele care se bazează pe puterea şi restricţiile conceptelor programarii orientate pe obiecte. Modelul restricţionează domeniul de aplicaţie al unei cereri la o clasă sau la o ierarhie de clase, avînd drept rădăcină acea clasă. Aceasta este o restricţie importantă, deoarece ea exclude cererile de tip join relaţional. Modelul permite, totuşi, utilizatorului folosirea grafului de clase pentru a specifica o cerere, predicatele putînd fi aplicate oricăror atribute ale claselor din graf. Această facilitate este similară extensiilor selecţiei relaţionale pentru relaţii imbricate. Faptul că, o cerere este adresată unei clase sau unei ierarhii de clase, avînd drept rădăcină o anumită clasă, este un lucru important, deoarece domeniul unui atribut este, în ultima instanţă,

o clasă specificată împreună cu toate subclasele sale directe sau indirecte. O problemă interesantă rămîne definirea unui model de cereri pentru bazele de date orientate pe obiecte care să admită operaţii echivalente cu reuniunea sau cu operaţiile pe mulţimi din sistemele relaţionale. De remarcat că, în timp ce în modelul relaţional rezultatul unei cereri este o relaţie, lucrurile nu sînt tot atît de simple în bazele de date orientate pe obiecte. În ciuda acestor diferenţe, modul de evaluare a cererilor în cele două modele este similar. Astfel, echivalentul operaţiei de selecţie din modelul relaţional revine la găsirea instanţelor unei clase ţintă, operaţie ce implică regăsirea recursivă a instanţelor altor clase referite drept valori de atribute. Regăsirea unei instanţe a clasei C2, care este valoarea atributului A a instanţei unei clase C1, este corespunzătoare operaţiei de reuniune a claselor C1 şi C2, avînd drept elemente comune atributul A din clasa C1 şi identificatorul definit de sistem al instanţei din clasa C2.

În bazele de date orientate pe obiecte, la baza indexării în memoria secundară, se găsesc ierarhiile de clase, ele fiind, de altfel, şi ţinta cererilor. Atributele moştenite sînt partajate în comun de către o clasă împreună cu toate subclasele ei directe sau indirecte. Din acest motiv, în locul definirii unui index pentru fiecare clasă din ierarhie, se recomandă folosirea unui index pentru combinaţia de atribute a tuturor claselor ce au drept rădăcină o clasă specificată de utilizator. Acest lucru este similar cu introducerea unui index suplimentar în sistemele relaţionale pentru atributele comune mai multor relaţii [24].

Tehnica indexării atributelor pentru o ierarhie de clase este denumită indexarea ierarhiei de clase, în timp ce metoda convenţională de indexare a atributelor în cadrul unei clase poartă numele de indexare a clasei. Din considerente de performanţă, ORION foloseşte indexarea ierarhiei de clase.

Indexarea atributelor imbricate este folosită pentru ierarhia compoziţională de clase. În acest caz, atributul indexat este un atribut indirect al clasei specificate. De exemplu, domeniul atributului "Fabricat" al clasei "Vehicol" este clasa "Companie", iar clasa "Companie" are atributul "Locaţie". "Locaţie" este un atribut imbricat al clasei "Vehicol". Indexul asociază valoarea unui atribut cu o listă de identificatori ai instanţelor unei clase. De exemplu, indexul "Piteşti" asociat atributului "Locaţie" al clasei "Vehicol" asociază o listă de vehicole fabricate de compania localizată în Piteşti (vehicole fabricate la Piteşti). Indexarea atributelor imbricate permite evaluarea cecerilor complexe într-un singur pas.

Actualizarea unui index al unui atribut imbricat este un proces complex şi costisitor. Ea se produce ori de cîte ori are loc un proces de înserare, ştergere sau actualizare al oricărei instanţe cuprinse în secvenţa de clase dintre locul unde se defineşte indexul şi cel unde este definit atributul.

Conceptele orientării pe obiecte impun redefinirea modelului de autorizare a accesului implementat pentru bazele de date relaţionale. Sistemul ORION este un exemplu în acest sens. Dreptul de a crea o subclasă este un tip de autorizaţie; ea este diferită de autorizaţia de a crea o simplă clasă, deoarece crearea unei subclase implică moştenirea atributelor şi metodelor din mai multe clase existente.

Un alt tip de autorizare este cel pentru crearea unui obiect imbricat, deoarece un obiect este, în general, o colecţie de obiecte legate prin relaţia de agregare. Cazul devine şi mai evident cînd modelul bazei de date orientat pe obiecte este extins cu versiuni. Un acelaşi obiect poate avea mai multe versiuni şi, drept urmare, autorizarea accesului la o versiune a sa devine importantă.

Un aspect interesant este acela de a trata autorizaţia drept o proprietate a unei clase. Acest lucru implică dreptul de a moşteni această proprietate de către toate subclasele unei clase. Este de dorit, totuşi, să existe şi posibilitatea ca utilizatorul să stabilească în mod explicit autorizaţiile la nivelul subclaselor.

Ierarhia de clase introduce o noua dimensiune în controlul concurenţei în sistemele de baze de date. În sistemele orientate pe obiecte, dacă atributele şi metodele sînt şterse sau adăugate unei clase, ele vor fi, de asemenea, şterse sau adăugate tuturor subclaselor sale. Aceasta înseamnă că, în timp ce o tranzacţie accesează o instanţă a unei clase, nici o altă tranzacţie nu are voie să modifice definiţia vreunei superclase a clasei respective. De asemenea, ţinînd cont că o cerere adresată unei clase implică evaluări, nu numai în acea clasă, ci şi în toate subclasele sale, ierarhia de subclase nu trebuie modificată de o altă tranzacţie pe durata evaluării cererii [13].

Reprezentarea internă a bazelor de date orientate pe obiecte este mai dificilă decît cea a sistemelor relaţionale, deoarece nu mai avem de-a face cu o simplă colecţie de relaţii independente, ci cu o colecţie de clase interconectate prin relaţii de generalizare şi agregare. Dacă ierarhia de clase este reprezentată astfel încît fiecare clasă să conţină numai atributele proprii ei (definite pentru ea), atunci prelucrarea unui mesaj adresat unei clase implică parcurgerea ierarhiei clasei respective pentru a identifica superclasa de la care sînt moştenite atributele sau metodele în cauză. Din aceste considerente, proiectanţii de sisteme orientate pe obiecte au "liniarizat" ierarhia de clase, astfel încît fiecare clasă să conţină informaţiile referitoare la toate atributele şi metodele (proprii sau moştenite), aplicabile ei.

Pentru regăsiri rapide este utilă gruparea (clustering) obiectelor aparţinînd aceleiaşi clase într-un singur segment continuu. Mai mult, ţinînd cont că un obiect imbricat conţine o serie de obiecte aparţinînd unor clase diferite, este recomandabil ca şi aceste obiecte să poată fi memorate într-un segment continuu, aşa cum se întîmplă în SQL/DS. Este de remarcat totuşi că, nu toate componentele unui obiect imnbricat sînt de egală importanţă în cadrul unei aplicaţii. De exemplu, este

utilă memorarea componentelor fizice ale obiectului "Vehicol" în același segment, dar nu și obiectul "Companie" pentru atributul "Fabricat" al obiectului "Vehicol". Altfel spus, componentele ce urmează a fi grupate trebuie specificate de către utilizator sub forma unui subgraf. Gruparea dinamică a obiectelor este costisitoare și, din această cauză, este, în general, evitată. Se poate întîmpla ca gruparea obiectelor imbricate să determine memorarea în segmente diferite a obiectelor care aparțin aceleiași clase.
Pornind de la aceste caracteristici de bază ale proiectării și implementării sistemelor de baze de date orientate pe obiecte, în secțiunea următoare vor fi analizate cîteva direcții majore ale cercetării în acest domeniu, care, prin rezultatele scontate, vor apropia sistemele realizate de un model teoretic și el în curs de definitivare.

6. Direcții de cercetare viitoare

Bazele de date orientate pe obiecte reprezintă un teren fertil pentru cercetări viitoare. Evidențiem cîteva direcții în acest sens.

Formalizarea. Pentru a putea dezvolta sisteme de baze de date, orientate pe obiecte este necesară formalizarea și/sau, cel puțin, standardizarea conceptelor orientării pe obiecte [9]. Dat fiind faptul că, importanța orientării pe obiecte este bazată pe conceptele de reutilizare și extensibilitate, viitoarele cercetări vor fi direcționate spre noțiunea de moștenire. Se încearcă, de asemenea, formalizarea unui limbaj de cereri, suficient de puternic pentru operații echivalente cu cele de reuniune și tratare a mulțimilor specifice modelului relațional.

Instrumente de proiectare a bazelor de date. Complexitatea modelului de date complică problemele proiectării logice și fizice a bazelor de date orientate pe obiecte, comparativ cu cele relaționale. Scopul urmărit este acela de a concepe instrumente de asistare a proiectării, cît mai prietenoase și eficiente. Cercetările privind evoluția sistemului ORION constituie un prim pas important în proiectarea logică a bazelor de date orientate pe obiecte. Cercetările curente privind structuri de memorare pentru baze de date relaționale non-1NF [12] sînt relevante pentru proiectarea fizică a bazelor de date orientate pe obiecte.

Optimizarea operațiilor frecvente. Există un număr de operații care au o frecvență foarte ridicată în cadrul sistemelor orientate pe obiecte. O astfel de operație este, de exemplu, trimiterea de mesaje. Sistemul trebuie să reacționeze foarte eficient în determinarea metodei corespunzătoare. De asemenea, sistemul trebuie să determine rapid locația fizică a unui obiect în cazul în care obiectul implicat se găsește pe disc, corespondența dintre identificatorul logic și adresa fizică trebuie făcută foarte repede.
Deși microprogramarea este o opțiune importantă în optimizarea operațiilor, ea are dezavantajul faptului că face sistemul mai puțin portabil.
Nucleu independent de aplicație. O abordare de perspectivă în construirea sistemelor de baze de date orientate pe obiecte constă în definirea unui nucleu de bază, al cărui model de date să fie suportat direct de sistem, și în adaptarea de către programatori a modelului aplicației la modelul acestui nucleu. Un astfel de nucleu standard nu a fost realizat pînă în momentul de față. Situația este asemănătoare și pentru celelalte modele de baze de date. De exemplu, modelul relațional nu poate capta anumite concepte semantice, cum ar fi generalizarea, și, în acest caz, utilizatorul este forțat să-și mapeze aplicația la primitivele modelului relațional. Ținînd cont de faptul că modelul orientat pe obiecte înglobează anumite concepte semantice, este de așteptat ca efortul de adaptare la nucleu să fie diminuat, comparativ cu cel al celorlalte modele de baze de date. Un exemplu este sistemul ORION al cărui nucleu nu conține noțiunea de metaclasă. O aplicație ORION, sistemul expert PROTEUS [3], necesită conceptul de metaclasă. Maparea metaclaselor la clasele sistemului ORION s-a dovedit a fi un proces relativ simplu. Implementarea directă a metaclaselor în ORION ar fi fost un proces mult mai costisitor.
O altă metodă interesantă de construire a sistemelor de baze de date orientate pe obiecte, constă în definirea unui subsistem corespunzător nivelului de memorare, și în folosirea lui drept nucleu pentru diferite modele orientate pe obiecte. În continuare, un nivel superior, cu corespondent direct în modelul orientat pe obiecte, poate fi construit peste acest nucleu. Astfel de eforturi se întreprind la Universitatea din Darmstadt cu sistemul DASDBS [26,28]. Eforturi similare se regăsesc și în proiectele O2 [5], IRIS și Observer pentru definirea unui model comun de memorare, care să poată fi folosit de diferite limbaje de programare, orientate pe obiecte.

Gestiunea obiectelor distribuite. Majoritatea sistemelor de baze de date orientate pe obiecte, ca de exemplu VBase, Statice, IRIS, GemStone, ORION și O2, au adoptat o arhitectură client/server. Într-un astfel de sistem, un server centralizat de obiecte gestionează întreaga bază de date persistente, iar fiecare utilizator are un spațiu privat de lucru, pe o mașină client, care conține copii ale obiectelor persistente. Un sistem client/server este însă un sistem de baze de date restricționat. Un sistem de baze de date orientat pe obiecte, complet distribuit, implică obiecte distribuite în cadrul unei rețele, distribuirea fizică a obiectelor fiind complet transparentă pentru utilizator. O versiune distribuită a sistemului ORION este în prezent operațională. De asemenea, prototipul AVANCE, realizat la Universitatea din Stockholm, a fost și el proiectat ca sistem de baze de date distribuit, orientat pe obiecte.
Tehnicile arhitecturale, necesare pentru construirea sistemelor de baze de date orientate pe obiecte

uniprocesor, trebuie extinse și îmbinate cu tehnologia bazelor de date relaționale distribuite [21]; acest obiectiv necesită eforturi considerabile și constituie un teren fertil de cercetare.

Modelare semantică. Modelul de date orientat pe obiecte, captează o serie de concepte ale modelului semantic. Totuși, noțiunile de versiune, obiect compus sau relație inversă nu sînt incluse. VBase și Iris permit folosirea relației inverse, în timp ce ORION și-a extins modelul de bază cu conceptele de versiune și obiect compus. Este de așteptat, în continuare, ca și alte concepte semantice să aibă un impact asupra arhitecturii sistemelor de baze de date orientate pe obiecte, inclusiv asupra evaluării cererilor, a structurilor de memorare și al controlului concurenței.

Facilități suplimentare. Bazele de date orientate pe obiecte și- au dovedit necesitatea în dezvoltarea sistemelor inteligente și de proiectare asistată de calculator. Aceste sisteme urmăresc un acces interactiv la o bază de date integrată, fapt ce implică modificări fundamentale în ceea ce privește noțiunile de "tranzacție" și "integritate a bazei de date". Modelul tranzacțional convențional este inacceptabil în sistemele CAD, deoarece durata tranzacțiilor se poate extinde la ordinul orelor și chiar al zilelor. Modelul convențional protejează fiecare tranzacție de efectul concurent al altor tranzacții. Tehnica zăvorîrii blochează accesul oricărei alte tranzacții, pe toată durata procesului. De asemenea, în cazul abordării unei tranzacții, toate actualizările produse de ea trebuie anulate. Acestea sînt principalele aspecte ale modelului convențional care, pentru tranzacții de lungă durată, îl fac inacceptabil.
Au apărut o serie de soluții pentru corectarea acestor deficiențe [4,19]. De exemplu, GemStone combină protocolul pesimist de zăvorîre cu unul optimist de control al concurenței. Totuși, sînt necesare în continuare cercetări pentru definirea unui model tranzacțional adecvat noilor cerințe, care să asigure integritatea bazei de date în condiții de eficiență acceptabile.

7. Concluzii

În ultimii cîțiva ani, au apărut cel puțin 12 prototipuri de sisteme de baze de date orientate pe obiecte, atît în medii universitare, cît și industriale: ORION (MCC Austin Texas), 02 (GIP ALTAIR), IRIS (Hewlett-Packard), EXODUS (University of Wisconsin at Madison), POSTGRES (University of California at Berkeley) etc. O serie de firme (Servio Logic, Object Databases, Object Design, Ontologic, Objectivity, Versant și Itasca) au lansat oferte comerciale pentru sisteme, de amploare și complexitate diferite, destinate în unele cazuri unor segmente înguste de piață. Totuși, pe lîngă avantajele fundamentale privind extinderea facilităților de prelucrare, de la tipurile de date clasice, la obiecte complexe (inclusiv multimedia), extinderea ariei de aplicabilitate a sistemelor de baze de date, de la domeniul economic, la domenii noi, de perspectivă (CAD/CAM, inginerie software, birotică), modelul de reprezentare natural, apropiat percepției utilizatorului, tehnologia bazelor de date orientate pe obiecte (și, în particular, sistemele respective) prezintă o serie de deficiențe între care: relativa lipsă de maturitate a modelului de date și, deci, a limbajelor asociate, absența rezultatelor unei abordări formale coerente și a unui standard, insuficienta atenție acordată dezvoltării de medii de programare.
În concluzie, domeniul este larg deschis cercetării, obiectivul fiind o nouă serie de proiecte care să depășească aceste deficiențe.

BIBLIOGRAFIE

1. ANDREWS, T., ș.a.: Combining Language and Database Advances in an Object-Oriented Environment. În: Proceedings of OOPSLA '87, 1987, pp.81-85.
2. ATKINSON, M.,ș.a.: The Object-Oriented Database System Manifesto, Altair Technical Report, no.30/89, 1989, pp.40-57.
3. BALLOU, N., ș.a.: Coupling an Expert System Shell with an Object-Oriented Database System, J. Object-Oriented Programming, vol.1, no.2, June/July, 1988, pp.17-26.
4. BANCILHON, F., KIM, W., KORTH, H.: A Model of CAD Transactions. În: Proc. Int. Conf. Very Large Data Bases, Stockholm, Sweden, Aug., 1985, pp.123-128.
5. BANCILHON, F.: Object-Oriented Database Systems. În: Proc. ACM SIGACT-SIGMOD Symp. Principles Databases Syst., Austin, TX, martie 1988, pp.237-241.
6. BANERJEE, J., ș.a.: Data Model Issues for Object-Oriented Applications, ACM Trans. Office Inform. Syst., ianuarie, 1987, pp.1-15.
7. BANERJEE, J., KIM, W., KIM, K., C.: Queries in Object-Oriented Databases. În: Proc. 4-th Int. Conf. Data Eng., Los Angeles, CA, februarie, 1988, pp.144-148.
8. BEECH, D.: A Foundation for Evolution from Relational to Object Databases. În: Proc. Conf. Extending Data Base Technol., Venice, Italy, aprilie, 1988, pp.321-325.
9. BEERI, C.: A Formal Approach to Object-Oriented Databases, Data and Knowledge Engineering, no.5, 1990, pp.29-37.
10. BERCARU, R., VOINEA, S., DRĂGAN, H., DUȚESCU, V.: Abordarea orientată pe obiecte în domeniul gestiunii bazelor de date, Raport INTELDATA, nr. 1/91, ICI, București, 1991.
11. BRETL, R., ș.a.: The GemStone Data Management System, in Object-Oriented Concepts, Applications

and Databases, MA: Addison-Wesley, 1989, pp.160-182.
12. CHEINEY, J.P., ş.a.: Un sous-système de gestion d'objets complexes pour un SGBD relationnel, TSI, vol.9, no.1, 1990, pp.11-20.
13. GARZA, J. F., KIM, W.: Transaction Management in an Object- Oriented Database System. În: Proc. ACM SIGMOD Int. Conf. Management Data, June, 1988, pp.233-237.
14. GOLDBERG, A., ş.a.: Smalltalk-80: The Language and its Implementation, Addison-Wesley, Reading, MA, 1983.
15. JORDAN, D.: Implementation Benefits of C++ Language Mechanisms, CACM, vol.33, No.9, septembrie, 1990, pp.41-50.
16. Khoshafian, S., ş.a.: Object Identity, Proceedings of OOPSLA'86, 1986, pp.159-166.
17. KIM, W., ş.a.: Architecture of the ORION Next-Generation Database System, IEEE Trans. on Knowledge and Data Engineering, vol.2, no.1, martie, 1990, pp.23-35.
18. KIM, W.: Object-Oriented Database Systems: Strengths and Weaknesses, J. Object-Oriented Programming, vol.4, no.4, iulie/august, 1991, pp.21-29.
19. KORTH, H., KIM, W/, Bancilhon, F.: On Long-Duration CAD Transactions, Inform. Sci., octombrie, 1988, pp.1-14.
20. LECLUSE, C., ş.a.: Modeling Generic Type and Method Inheritance, Raport ALTAIR No.19/88, 1988, pp.1-28.
21. LINDSAY, B. ş.a.: Notes on Distributed Databases. În: Distributed Data Bases, Cambridge University Press, Cambridge, MA, 1980.
22. MAIER, D., ş.a.: Development of an Object-Oriented DBMS. În: Proc. OOPSLA'86 Conf., Portland, OR., octombrie, 1986, pp.77-81.
23. MELLENDER, F., RIEGEL, S., STRAW, A.: Optimizing SmallTalk Message Performance. În: Object-Oriented Concepts, Applications and Databases, Reading MA: Addison-Wesley, Reading, MA, 1989, pp.124-148.
24. MISSIKOFF, M.: A Domain-Based Internal Schema for Relational Database Machines. În: Proc. ACM SIGMOD Int. Conf. Management Data, mai, 1982, pp.215-224.
25. PARSAYE, K., ş.a.: Intelligent Databases, John Wiley & Sons, New York, 1989, p.1-324.
26. PAUL, H., ş.a.: Architecture and Implementation of Darmstadt Database Kernel System. În: Proc. ACM SIGMOD Int. Conf. Management Data, mai, 1987, San Francisco, CA, 1987, pp.201-208.
27. ROWE, L., STONEBRAKER, M.: The POSTGRES Data Model. În: Proc. Int. Conf. Very Large Data Bases, Brighton, England, septembrie, 1987, pp.83-95.
28. SCHEK, H., ş.a.: The DASDBS Project: Objectives, Experiences and Future Prospects. În: IEEE Trans. on Knowledge and Data Engineering, vol.2, no. 1, martie, 1990, pp.34-43.
29. Skarra, A., Zdonik, S., Reiss, S.: An Object Server for an Object-Oriented Database. În: Proc. Int. Workshop Object-Oriented Database Syst., ACM/IEEE, 1986, pp.89-92.
30. STONEBRAKER, M., ş.a.: The Implementation of POSTGRES, IEEE Trans. on Knowledge and Data Engineering, vol.2, no. 1, martie, 1990, pp.18-29.
31. TELLO, E., R.: Object-Oriented Programming for Artificial Intelligence, Addison-Wesley, New York, 1989, p.1-155.
32. VELEZ, F., BERNARD, G., DARNIS, V.: The O2 Object Manager: An Overview. În: Proc. 15-th Int. Conf. Very Large Data Bases, Amsterdam, The Netherlands, North Holand, august, 1989, pp.254-258.
33. WEINREB, D., ş.a.: An Object-Oriented Database System to Support an Integrated Programming Environment, IEEE Database Eng., vol.11, no.2, iunie, 1988, pp.33-43.
34. WILKINSON, K., ş.a.: The IRIS Architecture and Implementation, IEEE Trans. on Knowledge and Data Engineering, vol.2, no.1, martie, 1990, pp.1-12.