

Pentru a înțelege a treia formă normală, trebuie să înțelegem mai întâi conceptul de dependență tranzitivă.

Despre un atribut care depinde de un atribut care nu este identificator unic (cheie primară) a relației se spune că *este dependent tranzitiv*.

Uitându-ne la relația Movie, observăm că atributul Genre_Description depinde de atributul Genre_Code, iar MPAA_Rating_Description depinde de MPAA_Rating_Code. Pericolul păstrării acestor descrieri în relația Movie este faptul că, în final, cele două attribute ajung să depindă de înregistrarea unui film, ceea ce duce la toate cele trei anomalii de date prezentate mai devreme în acest capitol.

Se spune că o relație este în *a treia formă normală* dacă îndeplinește următoarele două criterii:

- Relația este în a doua formă normală.
- Nu există dependențe tranzitive (cu alte cuvinte, toate attributele non-cheie depind *numai* de identificatorul unic).

Pentru a aduce la a treia formă normală o relație aflată în a doua formă normală, mutăm attributele dependente tranzitiv în relații în care depind numai de cheia primară

Avem grijă să lăsăm atributul de care depind acestea în relația originală, cu rolul de cheie externă. Va trebui apoi să reconstruim vizualizarea originală printr-o uniune. Ca efect secundar, toate attributele ușor de calculat sunt eliminate ca încălcări ale criteriilor celei de-a treia forme normale.

De exemplu, într-o bază de date pentru vânzări, Suma Totală este obținută înmulțind Cantitatea Cumpărată cu Prețul Unitar; așa cum se observă cu ușurință, Suma Totală este dependentă de Cantitatea Cumpărată și de Prețul Unitar. Presupunând că toate cele trei attribute sunt dependente de identificatorul unic al relației care le conține, este ușor de văzut că Suma Totală (rezultatul calculat) este, de fapt, *dependentă tranzitiv* de celelalte două attribute.

PROGRAMARE ORIENTATĂ OBIECT

Avantajele programării orientate obiect

(Șoavă, G., Programarea orientată obiect, Note de curs, pag. 9-10)

Orientarea spre obiecte aduce avantaje decisive cum ar fi: modelarea obiectelor aplicațiilor, modularitatea, reutilizabilitatea și extensibilitatea codului care conduc la o mai mare productivitate, și dezvoltarea unei mari calități a aplicațiilor.

Deoarece utilizatorii doresc ca programele lor să se comporte într-o manieră fiabilă și previzibilă, este important să se înțeleagă modul în care programarea orientată spre obiecte și condusă de evenimente va contribui la corecta structurare și modularizare a programului.

Într-o manieră globală, un program orientat spre obiecte este un ansamblu de obiecte care printr-un schimb de mesaje declanșează anumite operații sau metode, facilitându-le stările lor interne și returnându-le parametrii.

Prin crearea în mod vizual a unei singure linii de cod, se obține un program funcțional care, deși banal, demonstrează toate elementele unei corecte proiectări orientate spre obiecte și conduse de evenimente.

Conceperea este fără îndoială primul domeniu în care obiectele ușurează munca. Obiectele permit reprezentarea în mod direct a entităților lumii reale și a relațiilor dintre entități. Reprezentat în mod grafic cu ajutorul grafurilor, în care nodurile reprezintă clasele iar arcele legăturile generale sau de agregare a entităților, face posibilă vizualizarea unui model de aplicație deosebit de clar.

Se definesc concepte noi, ca tip, clasă, moștenire, obținându-se o însumare a avantajelor sistemelor de gestiune a bazelor de date, care interoghează eficient datele, și a limbajelor procedurale care permit calcule complexe.

În timp ce structura unei părți din variabile conține date și cealaltă funcții de tratare, se organizează programele în entități active compuse din structuri de date ce ascund modul de funcționare. Același nume de funcție poate fi utilizată pentru efectuarea de acțiuni similare la obiecte diferite, ceea ce permite constituirea unui limbaj abstract și prezentarea în maniere similare a obiectelor înainte diferențiate.

Avantajele utilizării programării orientată pe obiecte se pot sintetiza astfel:

Ușurința proiectării și reutilizării codului:

Odată ce este testată corectitudinea funcționării unor obiecte dintr-o aplicație, acestea vor putea fi folosite fără nici o problemă și în altă aplicație. Acest avantaj poate fi valorificat prin constituirea de biblioteci de obiecte. În ceea ce privește proiectarea, se facilitează descompunerea problemelor complexe în subprobleme simple, care pot fi ușor modelate cu ajutorul obiectelor (variabilele vor descrie proprietățile obiectelor modelate și metodele acțiunilor lor).

Abstractizare:

Proiectanții pot obține o imagine de ansamblu, urmărind comportarea obiectelor și interacțiunile dintre ele, detaliile fiind îngropate în compoziția obiectelor.

Siguranța datelor:

Abilitatea obiectelor de a se comporta ca niște “cutii negre”, de a putea fi folosite fără a se cunoaște compoziția lor, asigură confidențialitatea datelor utilizate și micșorează frecvența aparițiilor și efectul erorilor legate de manipularea greșită a tipurilor de date.

Modelând realitatea complexă, tehnicile orientate obiect, pun accentul pe comportamentul datelor, încapsulând în conceptul de obiect atât datele, cât și operațiile posibile asupra lor.

POO prezintă numeroase avantaje, printre care, alături de posibilitatea de reutilizare a modulelor de cod, se remarcă sporirea productivității și a modularității datorate câtorva caracteristici proprii programării orientate-obiect cum ar fi implementarea ascunsă a detaliilor, marcarea clară a granițelor dintre obiecte, comunicarea explicită între obiecte. De asemenea, este remarcabil faptul că nu este necesară cunoașterea intimă a unui obiect - adevărate "black boxes", obiectele au structura intimă ascunsă utilizatorilor (poți să pornești televizorul fără să fie nevoie să cunoști cum funcționează, poți să realizezi un montaj electronic fără să fie nevoie să cunoști detaliile constructive ale unui chip - este nevoie doar să cunoști relațiile dintre intrare și ieșire).

Programarea orientată-obiect constituie o bună metodă de organizare a programelor de calcul (software). Proprietățile POO conduc la un cod principal compact și elegant. Obiectele pot descrie mai bine conceptele pe care le reprezintă, fiind mai logice și intuitive decât modul tradițional, cu simple structuri de date.

Din punct de vedere educațional, aplicării POO conduce la formarea rapidă a unor concepte globale de funcționare a metodei elementului finit. Spre deosebire de metodele tradiționale folosite în învățământ, care pleacă de la o descriere detaliată a metodei, a conceptelor și a noțiunilor, cu construirea unui program de la scară mică la scară mare, programarea POO permite construcția unor aplicații prin asamblarea unor module existente, la nivel global. POO reprezintă astfel o metodă modernă, logică și eficientă nu numai în dezvoltarea de programe, dar și în utilizarea și înțelegerea acestora.

În programarea orientată obiect interfața este separată de implementare.

Interfața este partea vizibilă a clasei, parte care trebuie înțeleasă de utilizatorul acesteia. Implementarea este partea ascunsă, internă clasei, care este importantă doar pentru autorul clasei. Pot exista una sau mai multe implementări pentru o aceeași interfață. O implementare satisface cerințele unei interfețe dacă comportamentul definit de interfața este realizat de implementare.

Pe lângă avantajul simplificării, separarea aduce un plus de flexibilitate pentru implementatori, deoarece mai multe implementări pot servi o aceeași interfață. Implementările pot să difere în ceea ce privește eficiența de timp, spațiu, prețul sau calitatea documentației puse la dispoziție, sau orice ale caracteristici non-funcționale. De asemenea o singură implementare poate să satisfacă mai multe interfețe. În acest caz implementarea conține o uniune de metode cerute de fiecare din interfețe.

Numeroase limbaje de programare includ tehnici POO. În lucrarea de față se folosește, pentru exemplificare, limbajul Visual C++, deoarece C++ reprezintă limbajul standard în lumea POO, poate tocmai pentru că nu este un limbaj obiectual pur.

C++ permite ca programarea procedurală și cea orientată-obiect să fie folosite împreună într-un limbaj destul de uniform.

Limbajul C++ prezintă și alte avantaje față de celelalte limbaje obiectuale, cum ar fi:

- C++ produce un cod cu un timp de execuție foarte eficient;
- existența multor compilatoare, inclusiv a celor gratuite;
- posibilitatea utilizării modulelor existente Fortran și C;
- legăturile cu sistemul de operare Unix (datorate limbajului C);
- larga utilizare și acceptare a limbajului C++ a produs numeroase implementări, pe platforme diferite, existând multe biblioteci și programe de calcul reutilizabile;
- posibilitatea de interfațare cu numeroase programe importante (AutoCAD, Matlab,)

***Utilizarea bazelor de date, programarea OLE și COM
(Șoavă, G., Programarea orientată obiect, Note de curs, pag. 133-137)***

5.1. Utilizarea bazelor de date

Organizațiile cu profil economic acumulează o cantitate mare de informații în urma activităților comerciale de zi cu zi. Sunt înregistrate detalii despre clienți și furnizori, despre vânzări și necesarul de stocuri etc. Marea majoritate a corporațiilor optează pentru stocarea acestor date vitale în cadrul bazelor de date sau, mai exact, într-un **DMS (DataManagementSystem - sistem de gestionare a bazelor de date)**. Un sistem de gestionare a bazelor de date este un produs software care stochează datele într-o structură bine organizată și oferă mijloace eficiente de accesare și actualizare a acestor date.

Există două tipuri de baze de date: baze de date relaționale și baze de date **obiectuale**. Diferența dintre ele provine din modul conceptual în care sunt gestionate datele. Bazele de date relaționale au la bază tipuri de date simple, recunoscute (caractere, șiruri, întregi etc.) și nu permit crearea unor noi tipuri de date. Bazele de date obiectuale operează cu tipurile de date la un nivel mai înalt, permițând crearea prin definiție a unor noi tipuri de date. Aceste obiecte corespund celor create într-un limbaj orientat-obiect, de exemplu un obiect stocat într-o bază de date obiectuală ar putea fi o persoană sau un automobil.

Înainte de apariția sistemelor de gestiune a bazelor de date, aplicațiile scrise foloseau structuri de fișiere proprietare. Numai cei care proiectau și programau sistemul știau exact cum erau dispuse datele, ceea ce înseamnă că atunci când, utilizatorii sistemului aveau nevoie să manipuleze datele într-o manieră aparte, trebuiau de regulă să solicite o extindere a sistemului, ceea ce necesita timp.

Bazele de date relaționale au eliminat în bună parte această problemă Printr-o standardizare a modului de stocare a datelor, acestea fiind organizate pe tabele, coloane și linii.

Există mai mulți producători care oferă baze de date relaționale, fiecare bază de date are propria sa structură și, în consecință, un set propriu de funcții API. Din ce în ce mai mult se cere ca aplicațiile să poată lucra cu orice bază de date. Pentru a putea dezvolta aplicații care să utilizeze o bază de date relațională, indiferent de producătorul acesteia, este nevoie de o metodă generică de programare. O astfel de metodă există și se numește **ODBC (Open Database Connectivity - conectivitate deschisă a bazelor de date)**.

ODBC aduce o interfață de programare standard prin care pot fi utilizate diferite tipuri de baze de date relaționale. În acest scop este necesară existența unui intermediar care să traducă apelurile **ODBC** standard în apeluri de funcții specifice bazei de date. Acest intermediar este driverul **ODBC**, oferit de către producătorul bazei de date sau de către o firmă terță specializată în astfel de produse. **ODBC** a fost adoptat ca standard și astfel de drivere există acum pentru toate bazele de date răspândite.

Procedura de instalare a unor aplicații Microsoft, precum Office și Visual Studio, permite instalarea celor mai folosite drivere **ODBC** produse de Microsoft.

Comenzile sunt transmise driverului **ODBC** și pasate mai departe bazei de date cu ajutorul **SQL** (limbaj structurat de interogare). Acest limbaj a fost dezvoltat anume pentru accesarea bazelor de date și este acum standardul de facto. Există aproape tot atâtea variante de **SQL** câte baze de date sunt, fiecare producător aducând propriile îmbunătățiri. Dar există cerințe minime care asigură un set de comenzi care nu sunt standard, deoarece **ODBC** oferă o metodă de scurt-circuitare ce permite execuția directă a instrucțiunilor **SQL**. Utilizarea unor comenzi în afara standardului înseamnă ca aplicația să-și piardă capacitatea de a accesa bazele de date ale altor producători, tocmai această capacitate fiind avantajul major adus de **ODBC**.

Limbajul structurat de interogare **SQL** conține trei tipuri de comenzi: DDL, DML și DCL. Comenzile DDL (Data Definition Language - limbaj pentru definirea datelor) sunt folosite *pentru crearea și modificarea schemelor de baze de date*. Comenzile DML (Data Manipulation Language - limbaj pentru manipularea datelor) sunt folosite *pentru interogarea și manipularea informațiilor*. Comenzile DCL (Data Control Language - limbaj pentru controlul datelor) sunt folosite *pentru acordarea unor drepturi de acces specifice diferiților utilizatori*.

Prima sarcină legată de utilizarea **ODBC** este configurarea unei surse de date. O sursă de date indică programului unde se află fișierele care conțin baza de date și ce driver **ODBC** trebuie folosit pentru interpretarea administratorului **ODBC** pentru sursele de date, accesibil din panoul de control.

5.2. Noțiuni de programare OLE și COM

Complexitatea crescândă a dezvoltării aplicațiilor în cadrul unor medii complexe, așa cum este Windows, cu multele sale interfețe de programare a aplicațiilor (API), precum și nevoia de standardizare, de control al versiunilor, de accelerare a dezvoltării și de calcul distribuit, au dus la apariția unei tehnologii numite **programare bazată pe componente**.

Următorul pas în evoluția **COM** va fi **COM +**, Microsoft promite că acesta va simplifica în bună măsură scrierea codului necesar pentru **COM**, astfel că obiectele **COM** vor avea un comportament mai similar cu obiectele C++ și vor putea fi create prin intermediul cuvintelor cheie **new** și **delete** din C++.

Componentele sunt entități software de dimensiuni mici (asemeni instanțelor unei clase) care efectuează operații specifice prin intermediul unor interfețe bine definite. Spre deosebire de instanțele claselor, componentele nu sunt legate definitiv de o instanță a unui program sau de un sistem gazdă, pot fi scrise într-o multitudine de limbaje, și cu toate acestea, comunică fără probleme, prin intermediul interfețelor, cu programe și componente implementate în alte limbaje.

Microsoft a dezvoltat această tehnologie până la forma actuală, fiind denumită în prezent **Component Object Model (COM) - modelul componentelor obiectuale**. Se pot întâlni și alți temeni înrudiți, precum **legarea și încapsularea obiectelor (Object Linking and Embedding-OLE)** sau **controale ActiveX**. Acestea reprezintă implementări particulare ale programării **COM**. Propriu-zis **COM** este un **standard independent de limbaj și de platformă care definește modul în care obiectele pot comunica între ele prin intermediul unui protocol acceptat în comun**.

Cel mai important element privind obiectele **COM** îl reprezintă interfețele acestora; obiectele propriu-zise nu sunt decât cutii negre care implementează o funcționalitate particulară. Pentru a putea să se utilizeze această funcționalitate, programele trebuie să respecte un contract bine definit privind transmiterea parametrilor și obținerea rezultatelor. Acest contract dintre programul client și obiect se numește **interfață**.

Întregi biblioteci **API** pot fi definite și proiectate în termeni de interfață. Dacă se dezvoltă o aplicație client, se poate scrie programul astfel încât să comunice cu un server prin intermediul acestor interfețe acceptate, fără a mai conta ce aplicație server va fi utilizată. Reciproc, se poate decide dezvoltarea unor componente de tip server care respectă definițiile interfețelor, comercializându-le ca alternativă viabilă la componentele altor producători.

Interfața de programare a aplicațiilor pentru mesagerie reprezintă un set de interfețe standard pentru obiectele **COM**. Oricine este liber să scrie obiecte **COM** care efectuează operații precum *stocarea mesajelor, transportul mesajelor și oferirea către destinatarii mesajelor a unei agende de adrese*.

Microsoft Exchange este o implementare particulară a acestor componente server, dar există multe alte implementări. Codul efectiv s-ar putea să difere enorm între diferite implementări, dar toate obiectele **COM** respectă aceleași specificații de interfață. Aceasta înseamnă că, toți clienții acestor servicii, **Microsoft Outlook**, pot folosi componentele compatibile **MAPI** ale oricărui producător în scopul trimiterii, primirii și stocării mesajelor. În mod similar, orice producător poate să ofere propriile programe client (și mulți o fac) care folosesc **Exchange** sau orice alte componente server **MAPI**, chiar fără a ști ale cui sunt componentele utilizate. Singura cerință în ceea ce privește componentele client și server este ca acestea să se apeleze reciproc prin intermediul acestor interfețe definite de comun acord, reunite sub numele de **MAPI**.

5.2.1. Interfețe **COM**

O interfață este o definiție a unei mulțimi de funcții și a parametrilor acestora. Orice obiect **COM** dispune de cel puțin o interfață, iar în mod frecvent sunt oferite mai multe interfețe, fiecare conținând propriul său set unic de funcții.

Un obiect **COM** poate fi scris în orice limbaj care are capacitatea de a suporta aceste interfețe. Unele limbaje sunt mai potrivite în acest sens decât altele, de exemplu **JAVA** este foarte potrivit, din cauză că, fiecare obiect **JAVA** poate avea mai multe interfețe, deci se mapează natural peste un obiect **COM**. Obiectele **COM** conțin codul efectiv aflat în spatele acestor interfețe, astfel că un program care apelează o funcție declarată într-o interfață va găsi o implementare ce efectuează operația definită de acea funcție în cadrul interfeței respective.

Structura interfeței **COM** standard este exact aceeași cu structura tabelii de funcții virtuale din **Visual C++**, ceea ce înseamnă că este posibil să se folosească mecanismul tabelilor de funcții virtuale pentru a defini și implementa interfețe **COM**.

În mod normal, funcțiile virtuale sunt folosite ca o metodă de supradefinire în cadrul unei clase derivare a unei funcții din clasa de bază. În acest scop este declarată o tabelă de funcții virtuale asociată clasei respective.

Orice instanță de memorie a unui obiect **C++** are atașată o tabelă de funcții virtuale (chiar dacă se poate ca unele tabele să nu conțină nici o intrare și deci, să fie vide). Fiecare intrare din tabelă conține un pointer către codul care implementează o funcție virtuală. De fiecare dată când se apelează una dintre funcțiile virtuale ale obiectului, tabela atașată oferă adresa corectă, corespunzătoare fie funcției din clasa de bază, fie funcției din clasa derivată.

Declararea unei clase **C++** care conține exclusiv funcții virtuale pure, se numește **clasă de bază abstractă**. Nu este posibilă instanțierea unei clase abstracte, dar aceste clase se pot utiliza pentru crearea unei definiții de interfață **COM** compatibilă cu **C++**.

La crearea unei instanțe a unui obiect **COM**, un proces server aflat în rulare se va ocupa de inițializarea și gestionarea acesteia. Procesul server poate fi propriul program client, un **DLL** atașat, un executabil (.exe) aflat pe propriul calculator sau un program aflat pe un sistem la distanță. Fiecare din aceste ipostaze diferite se numește **context**.

Când se apelează funcții pe baza acestui pointer la interfață, propriul client și obiectul **COM** sunt puse în legătură cu ajutorul unui DLL terț, care se numește **de intermediere**, tehnica purtând numele de **apel intermediat**.

Biblioteca de intermediere are în sarcină *împachetarea parametrilor într-un format independent de mașină, în scopul transmisiei*. Este posibil apoi, să folosească apelurile de procedură la distanță pentru a apela o funcție a unui obiect **COM** aflat la distanță. Bibliotecile de intermediere pot fi generate automat, prin crearea definițiilor în limbajul de definire a interfețelor (IDL) și prelucrarea acestora de către compilatorul Microsoft IDL.

Vechea problemă a întreținerii versiunii corecte a aplicației este rezolvată printr-o simplă regulă. Odată ce s-a eliberat un obiect **COM** pentru a fi folosit de utilizatorii de pretutindeni, versiunile ulterioare trebuie să păstreze ne-alterate interfețele originale. Aceasta înseamnă că noile versiuni ale unui obiect **COM** trebuie să implementeze interfețe suplimentare, fără a le modifica pe cele originale. Programele client mai vechi, care nu știu să utilizeze decât interfața mai veche, vor solicita în continuare această interfață, în timp ce programele client care cunosc noile facilități pot să solicite versiunea mai nouă de interfață.

Prin convenție, numele noilor interfețe se creează prin adăugarea la numele original a numărului de versiune.

5.2.2. Automatizarea OLE

Legarea și încapsularea obiectelor (**OLE**) este un termen folosit inițial pentru a *descrie capacitatea de a insera obiecte de diferite tipuri în documente având un tip propriu*, un exemplu fiind inserarea foilor de calcul Excel în documentele Word.

Documentele care suportă operațiile de legare și încapsulare se numesc **documente compuse**. Obiectele inserate pot fi încapsulate, ceea ce înseamnă că pot fi salvate odată cu documentul, sau legate, ceea ce înseamnă că în document va fi inserată o referință la alt fișier (numită identificator). Acestea sunt cele două operații care au dat naștere termenului **OLE** original.

Semnificația acestui termen, s-a lărgit însă, incluzând acum **tragerea și plasarea OLE** și **automatizarea OLE**, acestea din urmă referindu-se la situația în care *un program poate să apeleze metodele unui alt program care rulează ascuns, fără ca utilizatorul să fie conștient de această interacțiune*.

Un obiect **COM** care oferă o interfață dispecer conține o tabelă de metode (funcții), evenimente (funcții care utilizează apeluri inverse ale clientului în scopul unei notificări a acestuia) și proprietăți (funcții care furnizează sau stabilesc valoarea unei anumite variabile a obiectului **COM**). Programele client pot să acceseze aceste informații dinamic (procedeu se numește legare târzie), în timpul rulării, pentru a afla detalii privind obiectul de automatizare.

Se poate întâlni termenul de interfață duală folosit în legătură cu **COM** și **OLE**. Un obiect **COM** poate să implementeze o interfață duală, oferind funcțiile atât printr-o interfață directă **COM**, cât și printr-o interfață dispecer. Astfel, programele client pot beneficia de avantajele ambelor soluții; programele C++ rapide pot să acceseze obiectul direct prin interfața **COM** rapidă, iar Visual Basic și limbajele de scriptare pot să acceseze funcțiile prin interfața dispecer mai înceată.

5.2.3. Crearea unui server propriu de automatizare OLE

Multe aplicații, printre care Word, Excel sau chiar Developer Studio, sunt **servere de automatizare**. Acestea pun la dispoziția aplicațiilor client o interfață dispecer care poate fi utilizată pentru apelarea funcțiilor proprii, oferind servicii specializate precum verificarea ortografică.

Visual Basic Scripting, instrumentul utilizat pentru crearea macro-comenzilor, folosește aceste metode în scopul creării și manipulării documentelor din cadrul serverelor de automatizare. De fiecare dată când se scrie o macro-comandă pentru unul din aceste programe, se folosește o versiune redusă de Visual Basic care permite apelarea funcțiilor din interfața dispecer a serverului de automatizare respectiv.

Serverele de automatizare sunt asociate de obicei cu un nume inteligibil care poate fi utilizat pentru determinarea identificatorilor de clasă corespunzători, care sunt reținute într-un registru intern, conținând subchei ce specifică identificatorul de clasă al serverului de automatizare respectiv.

Infrastructura unei aplicații server de automatizare poate fi creată prin intermediul AppWizard, validând opțiunea **Automation** din pasul trei din MFC AppWizard.

5.2.4. Servere și containere OLE

Un server **OLE** este o aplicație care poate fi lansată în cadrul unei ferestre a unei aplicații container. De exemplu, foile de calcul Excel, pot fi inserate și editate în Word. În acest caz, Word reprezintă aplicația container **OLE**, iar Excel este aplicația server **OLE**.

Se mai întâlnește termenul de **document activ**, folosit în legătură cu serverele și containerele **OLE**, ceea ce este o terminologie relativ nouă, legate de **ActiveX**, care dezvoltă **arhitectura container/server OLE** pentru a permite obiectelor încapsulate să câștige controlul asupra întregii zone client a containerului (nu doar asupra unui mic cadru) și să manipuleze direct fereastra, meniurile și barele cu instrumente ale acestuia.

Aplicațiile pot fi concomitent containere **OLE** și servere **OLE**, caz în care pot juca ambele roluri. Word-ul și Excel-ul sunt exemple potrivite în acest sens, deoarece se pot rula Excel și să se insereze documente Word și viceversa.

Un server complet poate să ruleze ca aplicație de sine stătătoare sau ca obiect inserat, în timp ce mini-serverele pot fi lansate doar din cadrul unui program container.

La inserarea unui obiect server **OLE**, acesta poate fi editat în interiorul unei suprafețe rectangulare care este mărginită de un chenar gros, hașurat, de culoare neagră, numit cadru local. Operația se numește **activare locală** și este inițiată prin transmiterea unor indicatori, numiți verbe, de la container către server. Atunci când nu este activ, cadrul dispare și imaginea afișată este ultima generată atunci când acesta era activ (stocată și reprodușă pe baza unui context dispozitiv de tip metafizier). Atunci când este activ, cadrul este afișat, iar la meniul propriu al container-ului sunt adăugate elemente de meniu ale obiectului server.

Datele documentului încapsulat pot fi serializate de către documentul container prin intermediul funcțiilor obișnuite de serializare.

Dacă un document încapsulat este stocat integral în cadrul documentului compus al aplicației client, obiectele legate sunt stocate sub forma unor simpli identificatori. Un astfel de identificator este un mic obiect care identifică în mod unic locația datelor propriu-zise și modul de afișare a acestora în cadrul aplicației client.

Infrastructura standard pentru servere **OLE** și containere **OLE** poate fi creată cu ajutorul AppWizard, însă cu toate acestea, între container și server există o cooperare complexă care trebuie implementată pentru a asigura funcționarea corespunzătoare a ambelor părți.

Prin personalizarea a două clase adăugate pe partea de server și a clasei suplimentare a containerului se poate implementa suport pentru operații destul de complexe de editare locală la rularea într-un cadru în fereastra unei aplicații container. Nici serverul și nici containerul nu au voie să cunoască tipul obiectului care încapsulează sau al celui conținut. Atât timp cât obiectele server și client respectă același standard, containerul și serverul pot fi integrate perfect pentru a da utilizatorului impresia unei aplicații unitare.