Proposed Framework which use the Object Oriented Principles in Relational Systems. General Aspects and Principles (Part I)

Lect. Cătălin STRÎMBEI
Catedra de Informatică Economică, FEAA, Universitatea „Al. I. Cuza" Iaşi

There are some significant theoretical and technological approaches on the issue of object-relational "impedance mismatch" between applications' abstract model and database structures. Two characteristics of those approaches we think that are questionable: first of all it is so called "flat" nature of relational systems and model, and then there is the drawback of the storage of (object oriented) semantics on the application level, thus severe limiting the data (object) sharing and, at the same time, virtually broking the data independence principle of database systems architecture. In this paper we will try to outline an approach to address to some in a concrete manner.
Keywords: *relational data structures design, object oriented principles, SQL3, UML, MDA.*

1 Caracteristicile abordărilor obiectual-relaţionale actuale

Clasic, obţinerea unei scheme de structurare a obiectelor într-o schemă de structuri SQL, considerată de nivel logic, se bazează pe un proces de mapare *obiectual-relaţional*, considerat necesar pentru a depăşi problemele de „incompatibilitate" dintre cele două paradigme de modelare (orientat obiect şi relaţional) şi dintre cele două niveluri de structurare a datelor (conceptual şi logic). Cele două modele fiind considerate iniţial incompatibile, sau, în termeni mai „fundamentalişti", „irenconciliabile", respectivul proces este considerat un „rău necesar", iar pentru fluidizarea şi reducerea costurilor de dezvoltare în această privinţă s-au făcut eforturi atât din perspectiva producătorilor de sisteme de gestiune a bazelor de date („extinderi" orientate obiect, altoite pe sistemele SQL-relaţionale existente), cât şi din perspectiva dezvoltatorilor de aplicaţii, care au izolat „procesul" de mapare în instrumente separate, ajungându-se chiar la pseudo-limbaje de lucru cu respectivele supra-structuri de date în măsură să adapteze cele două „medii" considerate conceptual şi fizic diferite.

Din punct de vedere teoretic, abordările OR „clasice" propun un set de principii generale pentru a obţine o schemă *relaţională normalizată în a treia formă* pornind de la un model conceptual orientat obiect, sau cel puţin entitate-relaţie. Există însă multe compromisuri necesare pentru a implementa rezonabil concepte cum sunt *moştenirea (relaţiile de generalizare şi specializare)*, *domeniile enumerative* şi *structurate, asociaţiile avansate cu atribute proprii* etc. Toate acestea formează ceea ce este cunoscut ca „impedance mismatch"[1], lucru care constituie de fapt sursa evoluţiei sau revoluţiei în transformarea modelului relaţional.

Abordarea considerată „ideală" în legătură cu structurile de date reflectând obiecte este surprinsă în sintagma *persistenţa obiectuală*. De fapt, principala caracteristică care derivă din acest concept se referă la *transparenţa tehnicii de mapare,* care poate fi tradusă prin minimizarea structurilor suplimentare necesare în procesul de convertire a obiectelor în structuri caracteristice modelului logic al bazelor de date.

O *definiţie* coerentă şi concisă a persistenţei poate fi considerată următoarea: *un obiect persistent reprezintă un obiect care continuă să existe şi după încheierea timpului de execuţie al programului care manipulează acel obiect* (Budd, 2002, p.579). Prin urmare persistenţa ar fi de fapt capacitatea de a salva starea obiectelor pe un suport permanent şi de a le reconstitui în mod transparent în spaţiul de execuţie al aplicaţiilor.

Mediile de programare orientate obiect oferă implicit facilităţi de obţinere a persistenţei printr-un proces numit *serializarea obiecte-*

[1] conceptul de incompatibilitate

lor. În Java, spre exemplu, acest proces se realizează în principal marcând aceste obiecte ca fiind „serializabile", prin implementarea de către clasele lor a unei interfețe specifice, *java.io.Serializable*. Permanentizarea acestor obiecte într-un fişier este însă delegată unor clase specifice separate *ObjectOutputStream* (scrierea) şi *ObjectInputStream* (citirea). Prin urmare, lucrul efectiv cu fişierul sau structura de stocare este delegată altor clase care ascund detaliile fizice. Pentru aplicațiile complexe de întreprindere, în medii multiutilizator, concurente, distribuite, accesate de la distanță, prin browsere Web spre exemplu, acest model de persistență este total nesatisfăcător datorită lipsei mijloacelor implicite pentru gestionarea concurenței, tranzacțiilor etc. Acest tip de facilități sunt asigurate la cel mai bun nivel de performanță prin serverele de baze de date, iar „partea leului" în acest sens o dețin cele considerate (SQL)relaționale.

Prin urmare, din motive practice mai mult decât teoretice, calea de urmat ar fi asigurarea persistenței obiectelor într-o structură sau mediu de stocare SQL-relațional, problema cheie fiind ca acest lucru să se facă într-o manieră la fel de transparentă ca şi seriabilizarea obiectelor în fişiere obişnuite. Acest demers ar putea fi sintetizat prin obținerea unui gen de *seriabilizare relațională* a obiectelor gestionate în medii obişnuite orientate-obiect.

Pentru a prezenta succint doar trei caracteristici esențiale, considerate determinante în ceea ce priveşte incompatibilitatea existentă între mediile orientate obiect de dezvoltare a aplicațiilor şi structurile de stocare relaționale, putem sublinia (Ambler, 2000b):

- *accesul* la obiecte/date: paradigma orientată obiect presupune „traversarea" obiectelor prin intermediul referințelor dintre acestea, iar paradigma relațională presupune reunirea structurilor separate (tabelelor) prin intermediul valorilor atributelor cu rol de chei străine;
- *identitatea*: paradigma orientată obiect consideră identitatea obiectelor ca fiind independentă de valorile atributelor, paradigma relațională ține cont de cheile primare (eventual compuse) construite pe baza valorii câmpurilor;
- *ierarhiile*: paradigma orientat obiectă consideră ierarhiile (de moştenire sau de agregare) ca fiind o cale naturală de construire a modelelor, pe când paradigma relațională se consideră că aplatizează ierarhiile, rezultând o serie de structuri relaționale intermediare, care nu reflectă în mod direct obiecte din lumea reală, ci aspecte particulare ale legăturilor (de asociere multiplă, de generalizare, de compunere etc.) dintre acestea.

Aceste considerente stau la baza contestării modelului relațional ca urmare a „limitării" sale semantice. În practică, în funcție de complexitatea aplicației (şi oarecum şi de abilitatea sau priceperea programatorului) există mai multe *abordări* în ce priveşte asigurarea (sub)stratului de persistență (Ambler, 2000a):

(1) Codificarea directă (hardcode) a dialogului SQL. În acest caz aveam de a face cu încapsularea directă în codul aplicației a codului sau apelurilor bibliotecii suport (API) pentru accesul la baza de date (gen JDBC).

(2) Crearea unor clase specifice care încapsulează funcționalitatea relațională corespunzătoare dialogului SQL. Această abordare presupune construirea claselor corespunzătoare suportului pentru persistență astfel încât să încapsuleze în metode specifice codul necesar comunicării (pe baza suportului JDBC) cu sursele de date. Aceste clase vor putea astfel să ascundă detaliile dialogului JDBC şi vor fi responsabile de persistența modificărilor instanțelor într-un mod cât mai transparent față de restul contextului aplicației.

(3) Crearea unui strat robust pentru persistență. Există mai multe tehnici şi tehnologii în acest sens, de la cele considerate mai rudimentare cum a r fi şablonul DAO (Data Acces Method), până la cele mai sofisticate care generează dinamic codul SQL şi gestionează (sau adaptează) inclusiv aspecte legate de concurență şi tranzacții, cum ar fi standardul JDO (Java Data Object) cu implementările aferente, cadrul de lucru open-source Hibernate care a inspirat standardul EJB 3.0.

Fără doar şi poate, aceste mecanisme au rezultat din eforturi de conceptualizare şi implementare remarcabile şi coerente față de

considerentele inițiale de la care s-au dezvoltat. Din punctul nostru de vedere însă, aceste abordări au cel puțin o insuficiență majoră: *portarea mecanismelor de persistență*, constând în logica generală de mapare, regulile de conversie şi limbajul de acces pentru respectivele structuri orientate obiect, la nivelul aplicației client. Ca urmare ***toată logica semantică de interpretare a datelor stocate nu mai este disponibilă la nivelul sistemului de gestiune a bazelor de date***. Ca urmare, structurile orientate obiect de organizare a datelor sunt practic nepartajabile la nivelul aplicațiilor ce lucrează cu SGBD-ul aferent bazei de date țintă. Rațiunea separării SGBD-urilor ca o clasă de sisteme separate o reprezintă tocmai necesitatea unificării viziunii şi standardizarea accesului pentru toate aplicațiile care procesează respectivele date, cu implicațiile privind integrarea sistemelor informaționale discutate în capitolul 2. **Acesta este principala motivație** pentru care propunem o *abordare alternativă* astfel încât *logica de organizare orientată obiect a datelor* să rămână la nivelul SGBD-ului şi astfel, indiferent de natura aplicațiilor (inclusiv platforma de dezvoltare), *partajarea datelor (obiectelor) să rămână efectivă*. În acest sens, înainte de dezvoltarea unui cadru de lucru metodologic am considerat necesară o fundamentare teoretică a acestei abordări propuse.

2. Implicarea ortogonală a principiilor orientate obiect în modelul relațional şi standardul SQL3

Ca o reacție la *Object-Oriented Database System Manifesto* (Atkinson et al., 1989), care propunea abandonarea eforturilor de structurare a datelor având la bază modelul relațional în favoarea unui model orientat obiect pur, C.J. Date şi Hugh Darwen lansează *The Third Manifesto* (Date&Darwen 1995 şi 2000), care se dorește o validare teoretică solidă a modelului relațional în contextul asimilării ortogonale a principiilor orientate obiect. În acest sens, cei doi autori recunosc *tipul* ca fundament al discursului privind structurarea datelor şi construiesc în acest sens un model de moştenire atât pentru tipurile simple, scalare, cât şi adaptat pentru tipurile compozite, adică TUPLU şi RELATION. În privința tradiției relaționale consacrate, Date şi Darwen fac o singură concesie care se dovedeşte totuşi esențială: clarifică conceptul de *atomicitate* în sensul diferențierii nete de cel de *scalabilitate*. Astfel, *scalabilitatea* se referă la distincția între tipurile *încapsulate* ale căror componente sunt eminamente interne, nefiind expuse direct, dar existând aşa-zisele *reprezentări posibile* destinate utilizatorilor, şi tipurile compozite sau structurate ale căror componente sunt vizibile şi accesibile neîngrădit de către sistem, materializate de fapt în tupluri şi relații. De asemenea, în cadrul unui tuplu sau relație, un atribut trebuie să fie *atomic*, adică va presupune *o singură valoare*, dar poate fi de orice tip, prin urmare inclusiv scalar sau structurat - tuplu, relație sau oricare alt tip, inclusiv colecție !

Noțiunea de tip (în special de tip *abstract*) se dovedeşte esențială şi în privința reformării standardului SQL şi a sistemelor relaționale comerciale. Astfel, standardul SQL3 urmează, în general, „caracteristicile critice" propuse de către Stonebraker (Stonebraker & Brown, 1999):

1. extinderea tipurilor de bază;
2. obiecte complexe (compozite sau înregistrări, seturi, referințe/pointeri de obiecte);
3. moştenire;
4. sistem de reguli active.

Tipurile complexe propuse în SQL-3 sunt ROW, SET, MULTISET, LIST. De menționat că tipul ROW se bazează pe noțiunea relațională de tuplu.

În *The Third Manifesto,* C.J. Date şi Hugh Darwen definesc un set extins de directive (26 relaționale, 6 orientate obiect, 25 pentru moştenire şi subtipizare), împreună cu un alt set de restricții sau interdicții (10 relaționale, 5 orientate obiect) şi un set de sugestii (10 relaționale, 5 orientate obiect), toate acestea constituind un model orientat obiect complet cu o solidă fundamentare pentru includerea caracteristicilor *tipurilor de date abstracte* şi fără definirea unor noi constructori revoluționari în modelul relațional inițial, dar aplicând cu rigurozitate principiile tipurilor asupra principiilor relaționale existente într-o manieră **ortogonală**.

Prin urmare, **tipurile** de date abstracte sunt folosite pentru a obține un cadru de lucru *consistent şi flexibil* pentru modelarea orientată obiect a sistemelor. Majoritatea limbajelor de programare orientate obiect oferă suport pentru tipuri folosind, de regulă, *clasele* ca un mijloc de definire implicit a funcționalității instanțelor tipurilor. Totuşi, noțiunea de tip ca specificație este formal mult mai bine reprezentată prin conceptul de *interfață*.

Clasele pot fi privite ca *tipuri* în principal pentru a obține un model orientat obiect a cărui coerență poate fi justificată formal prin intermediul *verificării predicatelor asociate tipurilor*. Deşi deciziile de proiectare conceptuală pot fi validate numai în practică, este totuşi destul de critică validarea relațiilor logice stabilite între constructorii conceptuali (care se pot regăsi ulterior şi în programare) pentru a evita inconsistențele ce pot apare în sistemul final. Iar acest lucru poate fi realizat doar prin intermediul tipurilor.

Tipurile pot fi privite drept *caracterizând* instanțele clasei corespunzătoare. Ele sunt prezente mai ales în domeniul *proiectării orientate obiect,* ca specificații ale claselor considerate drept un mijloc de implementare al tipurilor. Prin urmare, cel puțin în proiectarea OO, există o nuanță de diferențiere între tipuri şi clase: tipurile sunt considerate specificații, iar clasele sunt luate în considerare în implementare. Mai exact, *tipul* corespunzător unei clase este definit în domeniul proiectării OO pentru a defini: *scopul clasei, invariantul (restricția de bază) clasei, atributele clasei, operațiile clasei şi precondițiile, postcondițiile, definițiile şi semnăturile operațiilor* (Page-Jones, 2000).

Pentru a face o comparație preliminară între modalitățile de implementare a tipurilor abstracte prezentate anterior putem face următoarele considerații sintetice:

- În SQL-3 există mai multe categorii de tipuri: *tipurile fundamentale*, considerate ca atomice sau intrinseci (fără a avea componente vizibile utilizatorilor) şi extensibile prin intermediul *tipurilor distincte*, *tipurile de date abstracte* (expunând propriile componente sau atribute, plus identitate, moştenire, polimorfism) şi tipurile *colecții* (tipul ROW, LIST, SET şi MULTISET). Aceste categorii de tipuri sunt folosite în primul rând pentru a defini atributele din antetul tabelelor;
- În limbajul D (*The Third Manifesto*) tipurile sunt omniprezente, *scalare* sau *non-scalare (tupluri şi relații)* respectând principiile tipurilor abstracte: *împachetare* (tipurile scalare fără componente, dar având reprezentări posibile), componente definite de utilizator (tipurile non-scalare), moştenire şi substituabilitate (atât pentru cele scalare cât şi pentru cele non-scalare), fără *identitate explicită* însă.

Prin urmare *tipurile* pot fi considerate drept un concept de modelare comun, prezent în ambele câmpuri: *modelul conceptual* (orientat obiect) şi *modelul logic* (relațional). Dacă principiile tipurilor de date abstracte sunt aplicate de către ambele modele (conceptual şi logic), structura generală a sistemului va fi corectă, iar coerența sistemului modelat va fi atinsă într-un grad înalt dacă *aceleaşi tipuri vor fi reflectate în aceeaşi măsură în ambele modele.*

Date şi Darwen declară că principala *greşeală* care este făcută (şi care poate corupe inclusiv eforturile de modelare) este adoptarea *echivalenței între clase şi relații (tabele)* (Date&Darwen, 2000, p. 19). După cum am arătat mai înainte, clasele pot fi privite ca tipuri, iar echivalentul tipului în modelul relațional (clasic) este *domeniul* pe care Date şi Darwen l-au înlocuit pentru un plus de claritate cu ... tipul. Tabelele reprezintă într-adevăr *structuri de stocare*, şi, prin urmare, echivalentul corect ar fi un constructor legat de memorie adică *variabila* definită în contextul SGBD-ului (*variabila relație)*. Valorile stocate în acest tip de variabilă constau într-un set format din mai multe valori tuplu, având un antet corespunzător cu cel al variabilei relație. Prin urmare, prima „poruncă" este *să nu se pună semnul de egalitate între clasele din domeniul conceptual şi tabelele din modelul relațional* (am detaliat acestui subiect într-unul din paragrafele anterioare).

Principalul principiu de mapare poate fi formulat mai corect astfel: *clasele (domeniile sau tipurile) din modelul conceptual vor fi*

mapate pe tipurile existente în modelul relaţional (reconsiderat). Acest principiu clarifică în mare măsură problemele legate de „incompatibilitatea" obiectual-relaţional:

- *domeniile enumerative şi multi-valoare* pot fi reflectate prin *tipurile colecţie* (ROW, SET, LIST în SQL3 sau *tipurile generate* TUPLU sau RELAŢIE în D);
- *domeniile structurate* pot fi reflectate prin tipurile de date abstracte în SQL-3, sau tipurile scalare cu reprezentările externe posibile, sau tipurile non-scalare (cu propriile atribute definite în antet) generate TUPLU şi RELAŢIE;
- *generalizarea:* pe cât posibil, dacă respectă principiile subtipizării din teoria tipurilor de date abstracte, trebuie folosite tipurile TAD din SQL3, iar în D principiile moştenirii şi substituabilităţii au fost aplicate tuturor tipurilor (scalare şi non-scalare).

În legătură cu principiul echivalenţei obiectual-relaţionale, din viziunea propusă de către Date şi Darwen, enunţat prin ecuaţia clasă vs. tip, am putea propune *o altă viziune* pornind de la următoarea consideraţie: valorile stocate în structurile bazelor de date ar trebui să reflecte *stările* obiectelor şi, ca urmare, schimbarea stărilor obiectelor corespunde actualizării datelor din structurile bazelor de date. În consecinţă, obiectele sunt caracterizate prin *variabilitatea* stărilor care reprezintă valori aferente tipului declarat şi înscrise în spaţiul de stare eventual definit. Concluzia: cel mai bun echivalent pentru obiecte sunt *variabilele*, reflectate în modelul relaţional prin structuri de natura atributelor.

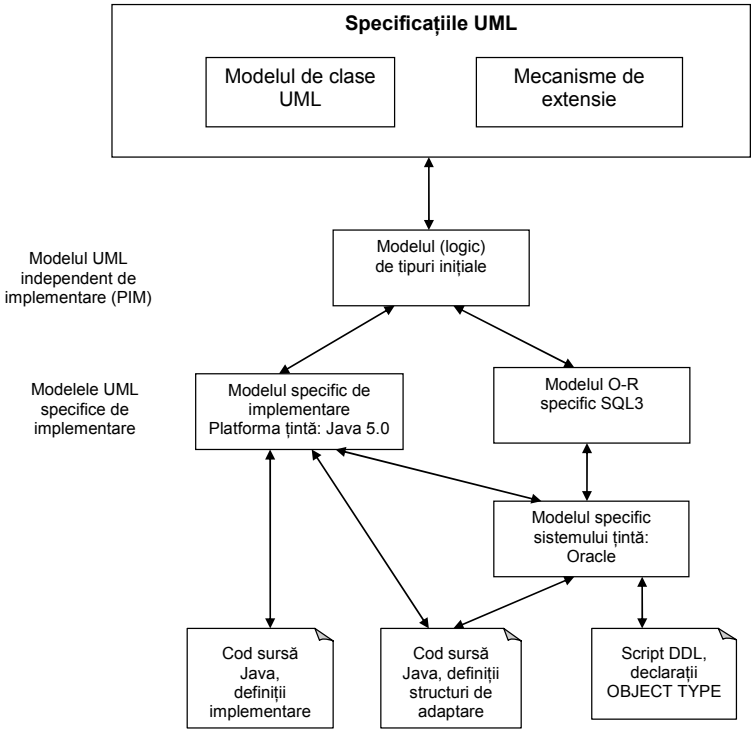

Fig. 1. Arhitectura MDA pentru cadrul de lucru propus

3. Obiectivele şi schema de principiu a cadrului de lucru MDA pentru implementarea unui model obiectual-relaţional

Obiectivele urmărite prin această propunere metodologică au în vedere:

- modelele dezvoltate astfel să fie cât mai „corecte" din punctul de vedere al logicii semantice, corespunzând astfel, cât mai exact, principiilor sistemului obiectual-relaţional ortogonal (vezi paragraful precedent);
- asigurarea portabilităţii integrale a *tipurilor de date* accesibile, disponibile sau partajabile prin intermediul sistemului de gestiune a bazei de date, dar fără de restricţii de implementare referitoare la platformă;
- reutilizarea pe cât mai mult şi mai liber posibil a logicii de implementare a tipurilor disponibile global la nivelul SGBD-ului.

În **concluzie,** îndeplinirea acestor obiective ar putea avea, după părerea noastră, alte câ-

teva consecințe importante, pe lângă soluționarea dilemei orientat-obiect vs. relațional, la nivelul sistemului de gestiune a structurilor de (obiecte)date:

- prezervarea semantică a schemelor de date (sau obiecte);
- ajustarea corespondenței sau alinierii între *modelul logicii afacerii* (nivelul organizațional) şi *modelul logic de organizare a structurilor de date (sau obiecte)*.

O schemă preliminară a arhitecturii MDA pe care o propunem în scopul atingerii obiectivelor menționate este prezentată în figura 1.

Descrierea structurală specifică a modului de abordare, ale cărui obiective le-am enunțat anterior, va fi realizată detaliat în următorul articol, unde vom prezenta şi principalele strategii de extindere a limbajului UML şi, de asemenea, regulile de mapare pentru interpretarea constructorilor propuşi.

Bibliografie

(Ambler, 2000a), Ambler, Scott W., *Mapping Objects To Relational Databases*, Ronin International – White Papers, Octombrie 2000 (http://www.ambysoft.com/mappingObjects.pdf)

(Ambler, 2000b), Ambler, Scott W., *The Design of a Robust Persistence Layer For Relational Databases*, Ronin International – White Papers, Noiembrie 2000 (http://www.ambysoft.com/persistenceLayer.pdf)

(Atkinson et al., 1989), Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S. *The Object-Oriented Database System Manifesto***,** în *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pag. 223-40, Kyoto, Japan, Decembrie 1989

(Date&Darwen, 2000), Date C.J., Darwen H., *Foundation for future database systems: the third manifesto: a detailed study of the type theory on relational model of data, including a comprehensive model of type inheritance* 2nd ed., Addison Wesley Longman, Inc. 2000

(Fortier, 1997), Fortier, Paul *SQL3. Implementing the SQL Foundation Standard*, McGraw-Hill, 1999