

Proposed Framework which Uses Object Oriented Principles in Relational Systems: Structure and Formating (Part 2)

Lect.dr. Cătălin STRÎMBEI

Catedra de Informatică Economică, Facultatea de Economie și Administrarea Afacerilor,
Univ. "Al. I. Cuza" Iași, linus@uaic.ro

Our approach tries to overcome the limitations of so called "flat nature" of relational systems, in the actual context of actual relational database theories, database systems technologies and object oriented methodologies by proposing an MDA framework to map an object oriented (UML formalized) model to object-relational structures of today's database systems.

Keywords: relational data structures design, object oriented principles, SQL3, UML, MDA.

1 Structura arhitecturii cadrului de lucru propus

În articolul precedent (Strîmbei, 2006) am încercat să prezentăm cât mai principial complexul problematic pentru care propunem o nouă abordare. În finalul respectivei lucrări am prezentat obiectivele avute în vedere, cu referire la corectitudinea semantică, portabilitatea și posibilitățile de reutilizare a modelelor obținute. De asemenea, am prezentat și o primă schiță de principiu a arhitecturii metodologice avute în vedere. Arhitectura de modelare multi-stratificată pe care am sugerat-o are la bază un experiment practic (Strîmbei, 2004), și presupune o implementare a șablonului MDA urmărind următoarele **niveluri de modelare**:

- modelul independent de implementare conținând *tipurile inițiale* ale structurilor de date;
- modelul specific mediului de implementare țintă (de exemplu Java) conținând, pe de o parte, constructorii abstracți (clase abstracte sau interfețe) definiți în contrapartidă cu tipurile inițiale plus, bine înțeles, constructorii (clasele) de implementare care să conțină definițiile atributelor și operațiilor necesare; acest model nu conține nici un fel de specificații sau dependențe față de sistemul țintă de gestiune a bazelor de date, sau de rezidență a structurilor de date, sistem care trebuie să asigure mediul de execuție necesar;
- modelul specific SQL, dar independent față de sistemul țintă, care să conțină echivalentul SQL3 al tipurilor inițiale;
- modelul de rezidență specific sistemului de

gestiune care să conțină, pe de o parte, definițiile SQL-dependente de sistemul țintă ale tipurilor SQL3, și, pe de altă parte, mecanismul de încorporare a constructorilor modelului specific mediului de implementare țintă în sistemul de gestiune al structurilor de date.

Modelul obiectual-relațional specific SQL3, introdus în sistemul de mai sus ca un gen de interfațare cu sistemul de baze de date țintă, este necesar funcție de (cel puțin) doi factori:

- adaptarea în sensul standardizării a unor constructori cât mai apropiați modelului relațional/ortogonal luat ca reper în paragraful anterior;
- relevanța acestuia în industria sistemelor de baze de date, care să impună în instrumentele de asistență gen CASE, sau în standardele aferente, un profil UML dedicat, coerent și complex în acest sens.

Altfel, ținând în continuare cont de existentele limite ale standardului SQL3 (de exemplu inexistența restricțiilor la nivel de tip), și de posibilitatea încorporării obiectivelor de armonizare orientată obiect ortogonală, rezultată din substanța actuală a modelului relațional, la nivelul cel mai independent din arhitectura de mai sus, modelul O-R specific SQL3 ar putea fi încorporat în modelul de tipuri inițiale prin adaptarea constructorilor cei mai apropiați la exigențele obiectual-ortogonale, iar în locul lui fiind adus direct modelul specific sistemului țintă.

Această opțiune este influențată de importanța *standardizării*, deocamdată alternativa cea mai convenabilă fiind prin intermediul limba-

jului SQL3, care este esențială, dat fiind obiectivul de portabilitate enunțat, pe două direcții (complementare):

- pe de o parte, posibilitatea implementării tipurilor inițiale pe orice platformă orientată obiect care asigură constructorii de abstractizare și implementare necesari;
- pe de altă parte, posibilitatea rezidenței și, mai ales, expunerii tipurilor (ca specificație și implementare) pe orice sisteme de baze de date care să asigure partajarea neutră a acestora.

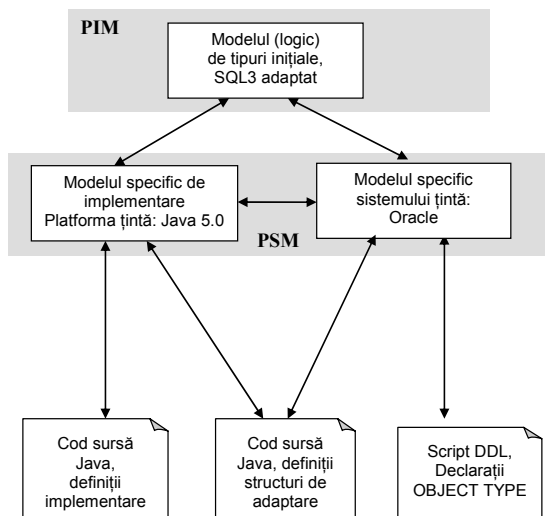


Fig.1. Modelul arhitectural al cadrului de lucru fără implicarea standardului SQL3

O a treia perspectivă asupra *portabilității*, fără legătură directă cu SQL, dar care, în contextul, *automatizării* procesului de obținere *materială* a modelului de implementare, se poate dovedi importantă, se referă la posibilitatea definirii, *încărcării* și modificării, și generării modelelor folosind *oricare* instrumente CASE care să îndeplinească un *număr minimal de cerințe*: să fie conforme UML (în privința constructorilor de bază și mecanismelor de extensie) și să permită codificarea modelelor în formatul standardizat MOF - XMI¹. Pornind de la abordarea din figura precedentă, putem deduce *trei etape de conversie*:

- conversia materială a modelului UML PIM

în modelul PSM al limbajului de programare țintă (Java) materializat în clase (Java) de implementare;

- conversia materială a modelului UML PIM în modelul PSM specific platformei sistemului de gestiune țintă (Oracle) materializat într-un script DDL (respectiv cu declarații CREATE [OBJECT] TYPE);
- conversia modelului UML PIM într-un model PSM de adaptare a claselor de implementare la mecanismul de găzduire specific sistemului de gestiune țintă (materializat în cadrul exemplului nostru concret în clase dependente de interfața *SQLDate*, din API-ul Oracle, și de clasele de implementare).

2. Specificații MDA necesare concretizării faptice a cadrului de lucru propus

Specificațiile de mapare avute în vedere ar trebui să se refere, minimal, la următoarele direcții:

- tipurile de bază simple: expunere, operatori, moștenire;
- tipurile compozite: expunere, operatori, moștenire;
- restricții;
- relații de asociere.

În privința tipurilor, propunem două **strategii de mapare**:

- tipurile modelate orientate obiect să fie mapate ca tipuri scalare (sau simple, necompozite), iar pentru acestea să fie definite colecții de tip relații, materializate de fapt în tabele distincte având o singură coloană de tipul asociat;
- tipurile compozite ca structură să fie mapate în tuplurile după care sunt definite relațiile, beneficiind astfel de întreg potențialul modelului relațional, și anume inferența tipurilor prin algebra relațională în primul rând.

Prima strategie este *cea mai directă* și cea mai aproape de *echivalența formală* în sensul modelului relațional cu structuri orientate obiect ortogonale. Este cea mai apropiată și de modele orientate obiect „pure”, gen ODMG, și este o manieră convenabilă de stocare a *stărilor* obiectelor, atât timp cât operațiile cu structurile din baza de date se limitează la accesul valorilor tipurilor cunoscute, fără a fi necesare operații suplimentare

¹ XMI – acronim pentru *XML MetaData Interchange*

de derivare semantică pentru a obține noi informații. A doua strategie este impură în sensul exact al definițiilor *tuplurilor* și *relațiilor* (tipuri și valori) din modelul relațional cu structuri orientate obiect ortogonale, însă aduce cele mai importante beneficii ca urmare a implicării funcționalității legate de operatorii specifici tuplurilor (inferența tuplurilor) și relațiilor (algebra relațională). O astfel de abordare a tuplurilor, prin existența în contrapartidă la nivelul sistemului a unor tipuri compozite implementabile extern, presupune însă un anumit grad de „impuritate”, generat de faptul că, prin excelență, tuplurile nu au (nu *trebuie* să aibă) o structură încapsulată, toate componentele fiind vizibile și manipulate prin operatorii de la nivelul generic al tipului TUPLU din care sunt generate toate celelalte, pe când proveniența lor externă implică o implementare distinctă, care ar putea însemna o structură internă, încapsulată. Găsirea unui compromis în acest sens aduce un avantaj important prin posibilitatea „portării” la nivelul sistemului de gestiune a semanticii tipurilor externe, materializată prin *operatori*, relații (asocieri generalizări) și chiar restricții (de tip) intrinseci. Compromisul minim care trebuie făcut presupune inexistența altor atribute (componente interne) decât cele declarate și accesibile public. Maniera de implementare cea mai la îndemână în acest sens este cea a *tabelelor tipizate*.

Pentru specificarea strategiilor de mapare ale **tipurilor**, propunem următoarele mecanisme UML (**nivelul logic al tipurilor inițiale**):

- stereotipuri diferite: <<TipScalar>> și <<TipCompozit>>, eventual specificate pornind de la definiția unui stereotip generic calificat prin <<TipDate>>;
- fiecare din aceste stereotipuri să beneficieze de o etichetă care să precizeze eventualul constructor SQL cel mai apropiat, sub numele *SpecificSQLConstruct*, valorile permise pentru această etichetă fiind șirurile de caractere (deci de tip CHAR) "TAD Scalar", pentru *TipScalar*, și {"ROW", "TAD Structurat"}, *TipCompozit*;
- mijloacele prin care sunt vizibile extern (echivalente componentelor reprezentărilor

posibile propuse de Date și Darwen) structurile împachetate intern ale tipurilor scalare să fie materializate prin elemente de model distincte, mai exact prin atribute stereotipizate <<ComponentaReprezentareTipScalar>>;

- componentele sau câmpurile tipurilor compozite să fie materializate prin elemente de model distincte, mai exact prin atribute stereotipizate <<AtributPentruTipCompozit>>.

Componentele reprezentărilor tipurilor scalare, la fel ca și atributele tipurilor compozite, pot fi de orice tip (scalar sau compozit).

Asociațiile între *tipurile scalare* pot fi modelate simplu (plan) prin intermediul componentelor din reprezentare. O altă strategie, favorizată de modelul relațional, implică asocierea valorilor de tip scalar în cadrul unor tupluri a căror structură definește antetul unor relații *clasice* care vor gestiona astfel respectivele asociații.

În cazul *tipurilor compozite*, asociațiile pot fi modelate eminentemente relațional „plat”, adică implicit prin intermediul cheilor străine și restricțiilor de integritate. O altă strategie, posibilă datorită clarificărilor actuale din modelul relațional, implică o abordare asemănătoare cu cea de mai sus: asocierea valorilor de tip compozit (tuplurilor în genere) în cadrul altor tupluri, pe baza cărora vor fi definite relații, ale căror atribute vor fi asociate tipurilor respectivelor valori compozite.

Pentru specificarea strategiilor de mapare ale **asocierilor** (sau *relaționărilor*), propunem următoarele mecanisme (**nivelul logic al tipurilor inițiale**):

- stereotipuri diferite pornind de la genurile diferite de tipuri asociate: <<AsocatieTipuriScalare>>, respectiv <<AsocatieTipuriCompozite>>, derivate din clasificatorul *asociație*;
- fiecare stereotip decorat cu eticheta *SpecificSQLConstruct*;
- pentru <<AsocatieTipuriScalare>>, *SpecificSQLConstruct* va lua valorile {"Atribut TAD Scalar", "ROW", "TAD Structurat"};
- pentru <<AsocatieTipuriCompozite>>, *SpecificSQLConstruct* va lua valorile {"FK", "ROW", "TAD Structurat"}.

Din punctul de vedere al relațiilor de generalizare care ar trebui să implice conceptul de **moștenire**, în privința tipurilor compozite considerăm că cel mai „curat” echivalent decurge din modul de inferență al tuplurilor implicat prin definirea relațiilor (sau tabelor) virtuale. *Moștenirea*, așa cum este în general acceptată, se referă la „preluarea” de către sub-tipuri a proprietăților sau caracteristicilor super-tipurilor. Acest fapt se manifestă, după părerea noastră, în două direcții:

- pe de o parte se manifestă structural: sunt moștenite atributele și asociațiile;
- pe de altă parte se manifestă semantic: operatorii și restricțiile.

Din perspectivă structurală, sub-tipurile ar trebui să *extindă* „definițiile” moștenite. Prin urmare, structura sub-tipurilor *derivă* din cea a tipurilor inițiale, operațiune care este cel mai bine echivalată în modelul relațional prin *derivarea* unui nou tip tuplu astfel: antetul acestuia ar fi format din cel puțin două secțiuni dintre care una trebuie să reflecte antetul tipului inițial la care se adaugă atributele suplimentare. Secțiunea moștenită ar putea lua forma unui *atribut de tip tuplu*, care ar putea fi eventual despachetată prin operatorul UNWRAP caracteristic tuplurilor din sistemul relațional actualizat de Date și Darwen (cu structuri orientate obiect ortogonale).

Din perspectivă semantică, modelul de moștenire propus relaționalului de către Date și Darwen absolutizează latura semantică, ignorând cu desăvârșire aspectele de implementare structurală. Din punctul lor de vedere, în cazul tipurilor tuplu și relație, structurile *sub* și *super* tipurilor trebuie să fie similare, adică să aibă același număr de atribute, iar tipul acestora să respecte principiul substituabilității: tipurile atributelor subtipului vor fi sub-tipuri pentru tipurile atributelor din super-tipuri (Date&Darwen, 2000, p.337). Prin urmare, în cazul tipurilor non scalare, sub-tipurile pot „rafina” prin subtipizare tipul atributelor „moștenite” și pot impune noi restricții, dar care să nu le „slăbească” pe cele de la nivelul super-tipurilor.

În consecință, și în cazul **moștenirii** propunem două strategii:

- *derivarea* structurală, care se manifestă în

cele din urmă prin tabele virtuale;

- *subtipizarea*, caz în care păstrăm simbolistica obișnuită.

Ca mecanism de reprezentare, specificarea **moștenirii** va porni de la relația de generalizare care în cazul tipurilor scalare va fi preluată ca atare, iar în cazul tipurilor compozite va fi *extinsă* prin intermediul a două stereotipuri:

- *derivare*, stereotip <<DerivareStructurală>>, care poate conduce în cele din urmă la tabele virtuale;
- *subtipizare*, stereotip <<SubTipizareSemantica>>, care va fi păstrată *doar ca legătură între tipuri*, și nu între tabele, așa cum se sugerează în standardul SQL3, nepreluând în acest sens nici conceptul de *tabele tipizate echivalente claselor instanțiate* (Gulutzan&Pelzer, 1999).

Aceste două strategii nu mai sunt însă opționale, ca în cazul asocierilor, ci restrictive conjunctural, funcție de relația structurală dintre tipurile relaționate.

Strategiile propuse de abordarea MDA pentru transformarea modelului independent (PIM) în modelul dependent de platformă (PSM) constau în principal în:

- interpretarea tipurilor originale ale PIM și maparea lor în elementele PSM aplicând setul de reguli specifice în acest sens;
- marcarea elementelor PIM instanțiate și maparea lor în elemente PSM aplicând un set de reguli parametrizate după respectivele marcaje;
- combinarea celor două strategii de mai sus. Aplicarea regulilor de mapare se va materializa, bineînțeles, atât în elemente constructive (tipuri), inclusiv structura lor, specifice platformei de implementare avut în vedere, cât și în caracteristici funcționale dependente de mediul de implementare.

Strategia pe o propunem pentru cadrul de lucru de față se bazează pe:

- formalizarea meta-modelului PIM prin intermediul unui sistem de stereotipuri specifice: <<TipScalar>>, <<TipCompozit>>, <<AsociatieTipuriScalare>>, <<AsociatieTipuriCompozite>>, <<DerivareStructurală>>,

<<SubTipizareSemantica>>,
 <<ComponentaReprezentareTipScalar>>,
 <<AtributPentruTipCompozit>>;
 ▪ marcarea corespondenței SQL prin intermediul unor *etichete* (tagged value) *SpecificSQLConstruct*, asociate fiecărui stereotip, dar restricționând valorile posibile la

specificul acestora.
 Regulile de transformare pot fi împărțite în:
 ▪ reguli parametrizate după sistemul de marcaje format din stereotipuri și etichete;
 ▪ reguli implicite deduse din arhitectura propusă a cadrului de lucru.

Tabulul 1. Reguli de transformare pentru cadru de lucru propus

Indicativ regulă	Marcaj PIM (parametri regulă: stereotip/etichetă):	Valoare	Șablon elemente generate în PSM
R1	<<TipScalar>> <i>SpecificSQLConstruct</i>	"TAD Scalar"	CREATE TYPE ... (...) AS OBJECT TYPE EXTERNAL NAME '...' LANGUAGE JAVA interface { ... } public class ... implements ... <interface> ... { ... }
R1.1	<<TipScalar>> cu <<ComponentaReprezentareTipScalar>> <i>SpecificSQLConstruct</i>	"TAD Scalar"	CREATE TYPE ... (...) AS OBJECT TYPE EXTERNAL NAME '...' LANGUAGE JAVA public abstract class ... { ... } public class ... extends ... <abstract class> ... { ... } public class... extends ...implements SQLData { ... }
R1.2	<<Tipcompozit>> cu << AtributPentruTipCompozit >> <i>SpecificSQLConstruct</i>	"TAD Structurat"	CREATE TYPE ... (...) AS OBJECT TYPE EXTERNAL NAME '...' LANGUAGE JAVA; CREATE TABLE ... AS ... <nume Tip Structurat> public abstract class { ... } public class ... extends ... <abstract class> ... { ... } public class... extends ...implements SQLData { ... }
R1.3	<<Tipcompozit>> cu << AtributPentruTipCompozit >> <i>SpecificSQLConstruct</i>	"ROW"	CREATE TYPE ... (...) AS OBJECT TYPE; CREATE TABLE ... AS ... <nume Tip Structurat> public abstract class { ... }
R2 R2.1	<<AsociatieTipuriScalare>> <i>SpecificSQLConstruct</i>	"ROW" sau "TAD Structurat"	CREATE TYPE ... (...) AS OBJECT TYPE; CREATE TABLE ... AS ... <nume Tip Structurat>
R2.2	<<AsociatieTipuriCompozite>> <i>SpecificSQLConstruct</i>	"ROW" sau "TAD Structurat"	CREATE TYPE ... (...) AS OBJECT TYPE; CREATE TABLE ... AS ... <nume Tip Structurat>
R2.3	<<AsociatieTipuriCompozite>> <i>SpecificSQLConstruct</i>	"FK"	<Preexistă: CREATE TYPE T1 ... (...) AS OBJECT TYPE; CREATE TABLE TT1 ... AS ... T1 CREATE TYPE T2 ... (...) AS OBJECT TYPE; > CREATE TABLE TT2 ... AS ... T2; ALTER TABLE TT2 ADD FOREIGN KEY ... REFERENCES TT1...)
R3	<<DerivareStructurală>>		<Preexistă:

	>		CREATE TYPE T1 ... (...) AS OBJECT TYPE; CREATE TABLE TT1 ... AS ... T1 > CREATE VIEW TT2 AS SELECT TT1.*, [<atribut nou>, ...] FROM TT1
R3.1	<<SubTipizareSemantica>> >		<Preexistă: CREATE TYPE T1 ... (...) AS OBJECT TYPE EXTERNAL NAME '...' LANGUAGE JAVA; > CREATE TYPE ... UNDER T1 AS OBJECT TYPE EXTERNAL NAME '...' LANGUAGE JAVA; /* fara clauza de specificare attribute suplimentare*/

Tabelul de mai sus prezintă doar „scheletul” de bază al cadrului de lucru propus. Rolul acestuia este să schițeze concepția și strategiile perspectivei sau abordării noastre privind implementarea relațională a unor structuri de date orientate obiect. Pe acest fundament se poate clădi în continuare pentru a obține un sistem de implementare *complet formalizat*, ținând cont de posibilitățile și potențialul sistemelor compatibile SQL3, dar și de cerința unui grad de echivalență cât mai bun față de principiile modelului relațional cu structuri orientate obiect ortogonale formalizat de către Date și Darwen.

3. Concluzii

Punctul de plecare al demersului sintetizat în aceste (două) articole îl constituie abordarea clasică a „incompatibilității” obiectual-relaționale cu mijloacele metodologice și tehnologice ce au calificat modelul relațional drept „plat”. Pe de altă parte, o altă caracteristică a soluțiilor (mai ales tehnologice) actuale presupune capturarea semanticii orientate obiect, cu care au fost „potențate” structurilor relaționale plate, la nivelul aplicațiilor și nu la nivelul sistemelor de baze de date, așa încât principiul independenței datelor în privința „obiectelor” devine inoperabil.

Cadrul de lucru pe care îl propunem își bazează infrastructura metodologică pe caracteristica de *genericitate* a limbajului UML care permite adaptarea modelului la considerațiile particulare ale domeniului avut în vedere, prin introducerea unor elemente suplimentare de meta-modelare (sau adaptare a celor existente) folosind mecanismele de extensie. Principiile de organizare a datelor

prin acest cadru de lucru considerăm că au un dublu avantaj față de abordările anterioare:

- pe de o parte, partajarea datelor, dar și a semanticii orientate obiect a lor, la nivelul sistemelor de baze de date, păstrând nealterat astfel principiul independenței (logice și fizice) a datelor;
- pe de altă parte, obiectivul de portabilitate (pe *trei direcții*: a mediului de implementare, a platformei de rezidență și a platformei de proiectare) avut în vedere ar trebui să asigure o productivitate confortabilă și un suport rezonabil de fezabilitate tehnică a soluției.

Bibliografie

- (Budd, 2002) - Budd, Timothy *An introduction to object-oriented programming Third Edition*, Addison Wesley, Pearson Education, Inc., 2002
- (Fortier, 1997) - Fortier, Paul *SQL3. Implementing the SQL Foundation Standard*, McGraw-Hill, 1999
- (Gulutzan&Pelzer, 1999) - Gulutzan, Peter, Pelzer, Trudy *SQL-99 Complete, Really: An Example-Based Reference Manual of the New Standard*, Miller Freeman, Inc., Lawrence, USA, 1999
- (Strîmbei, 2004) – Strîmbei, Cătălin *Object/Relational Impedance Mismatch. A Positive Approach And A Feasible With SQL And Java*, Vol. *The Proceedings Of The Central And East European Conference In Business Information Systems*
- (Strîmbei, 2006) - Proposed Framework which Uses the Object Oriented Principles in Relational Systems. General Aspects and Principles (Part I), revista Informatica Economică, A.S.E. București, nr.37/2006