Aspecte privind obtinerea unui strat de persistenta robust pentru aplicatiile obiectuale

Asist. Catalin STRÎMBEI
Facultatea de Economie si Administrarea Afacerilor, Universitatea „Al. I. Cuza" Iasi

Beyond the whole promotional "clamors" regarding the new platforms (J2EE, MS .NET) for enterprise-scale applications, we can say that actual technological "age" belongs to distributed (n-tier) technologies. On the logical level, the classical three layer approach (presentation, business, data) can be extended to six layer in attempt to cover in detail the specific aspects regarding, on the one hand, the syntax and context validation of data (services that are located in presentation layer) and, on the other side, the mapping (translation of) business request in the suitable SQL messages, plus data access services. The mapping services must be differentiated because at this level we need to solve the "impedance mismatch" problem between object-oriented business processing and relational structure of data storage. The solutions proposed for this problem can be quite simple (like direct coding of SQL dialog through the API calls of a specific library) but very inflexible, or very complex (like customizing or purchasing one "robust" persistence layer) but with an important gain in flexibility and effectiveness.

Keywords: *distributed architectures, object modeling, persistent object, object-relational impedance mismatch, object/relational mapping.*

Necesitatea constituirii unui strat de persistenta distinct în arhitectura apli-catiilor de întreprindere actuale

Restrictiile tinând de scalabilitatea hardware-ului fata de necesitatile de procesare în crestere, conectarea sistemelor dezvoltate anterior disparat în conditii de scalabilitate si eficienta, integrarea sistemelor dezvoltate pe baza mediilor suport provenind de la furnizori diferiti (de exemplu servere de baze de date diferite), restrictiile tinând de serviciile de securitate, tranzactii, integritate a datelor ce trebuie avute în vedere, toti acesti factori au împins software-ul de întreprindere catre o evolutie arhitecturala de la un singur strat (centralizare tip mainframe) la medii distribuite pe n-straturi (n-tier). Astfel ca, daca la început aspectele tinând de prezentare, logica afacerii si stocarea datelor erau practic nedelimitate, *sistemele client/server* (*pe doua straturi*) au realizat prima bresa preluând toate serviciile de prezentare si o parte din logica afacerii într-un strat distinct, dar introducând însa noi probleme tinând în special de administrarea aplicatiilor instalate pe multitudinea de clienti raspânditi în diverse locatii. Meritele sistemelor client/server se vadesc în special în arhitecturi multistrat în care descentralizarea prelucrarilor înseamna distribuirea aplicatiilor într-unul sau mai multe straturi intermediare, scalabilitatea, siguranta, eficienta, extensibilitatea dovedindu-se caracteristici esentiale pentru succesul aplicatiilor de întreprindere.

Arhitecturile pe trei straturi, fata de arhitecturile client/server pe doua straturi, separa logica afacerii pe un strat distinct localizat de regula pe un server de aplicatii care comunica strâns cu serverul de baze de date. Paradigma dezvoltarii software-ului bazat pe componente ce comunica între ele prin interfete standardizate a facut posibila apoi si cresterea numarului de straturi armonizate în arhitecturi generic numite *n-tier*.

În fapt acest strat suplimentar va gazdui asa numitele *business object*, care se regasesc într-o forma sau alta cam în toate tehnologiile în care ar putea fi implementata o astfel de solutie (COM+, .NET sau J2EE). Prin urmare arhitectura pe trei straturi se bazeaza pe capabilitatea de a dezvolta aplicatii partitionate, ceea ce presupune împartirea acestora în componente logice grupate în unul din urmatoarele straturi: serviciile utilizator (în

principal servicii de prezentare), servicii privind logica afacerii (business services) si servicii de date [Szabo98].

Arhitectura pe trei straturi pentru aplicatiile de întreprindere implica în general mai multe tipuri de servicii prezentate în tabelul 1.

Tabelul 1. Scopul si obiectivele într-o arhitectura pe trei straturi [Ahmed &Umrysh2002]

Strat	Scop	Obiective
Servicii de prezentare (Presentation Services)	Prezentare datelor Verificarea preliminara (acceptarea) datelor GUI (Graphical User Interface)	Usurinta în utilizare Interactiuni naturale, intuitive cu utilizatorul Timpi mici de raspuns
Servicii functionale ale afacerii (Business Services)	Regulile de afacere de baza Flux de control al aplicatiei/dialogurilor Aplicarea regulilor de integritate a datelor	Aplicare rigida a regulilor afacerii Conservarea investitiilor în cod Reducerea costurile de întretinere
Servicii de date (Data Services)	Stocarea durabila a datelor si regasirea lor Accesul SGBD-urilor prin API-uri specifice (ODBC, JDBC, ADO) Controlul concurentei	Baza de date consistenta, sigura si securizata Partajarea informatiei Timpi mici de raspuns

În fapt aceasta arhitectura pe trei straturi poate fi detaliata într-o *arhitectura pe 6 straturi logice*, descompunând stratul *Business Services* (serviciile pentru afacere) în doua:

- *Business Context Services* (serviciile de context) pentru filtrarea si „curatarea”/ verificarea informatiei provenita din interfata grafica;
- *Business Rule Services* (serviciile regulilor afacerii) pentru implementarea regulilor traditionale ale afacerii.

De asemenea stratul *Data Services* (serviciilor pentru date) poate fi descompus la rândul lui în:

- *Data Translation Services* (serviciile de traducere/interpretare a cererilor de date) pentru traducerea cererilor de informatii specifice „limbajului” stratului afacerii într-un limbaj compatibil cu serviciul de stocare a datelor (adica în SQL);
- *Data Access Services* (serviciile de acces la date), cererile rezultate din stratul anterior sunt executate prin intermediul unui API specific comunicarii cu SGBD-ul care gazduieste baza de date (un driver JDBC de exemplu);
- *Database Services* (serviciile corespunzatoare bazelor de date).

Prin urmare, conform arhitecturi pe 6 straturi comunicarea dintre stratul serviciilor functionale ale afacerii – *Business Services* si serviciile de date – *Data Services* ar trebui realizata prin intermediul unui strat intermediar *Servicii de traducere/interpretare a cererilor de date* (Translation Services). Aceasta abordare este luata în considerare de majoritatea dezvoltatorilor datorita naturii diferite a celor doua medii conceptuale si tehnologice (modelul orientat obiect al aplicatiei – de exemplu Java - prin care sunt specificate regulile afacerii si modelul relational prin care sunt formalizate structurile de stocare a datelor) care trebuie mapate prin intermediul acestui strat suplimentar.

Între *serviciile regulilor afacerii* a caror codificare cade în sarcina programatorului de aplicatii si *serviciile corespunzatoare bazelor de date*, care cad în principal în sarcina administratorului bazei de date, se gasesc cele doua straturi de care depinde în mod esential asigurarea într-o maniera cât mai transparenta a persistentei datelor (sau mai exact a persistentei obiectelor): serviciile de translatare/mapare si serviciile de acces la date.

Aceste doua tipuri de servicii pot lua mai multe forme în natura lor functie de *tipul mediului de stocare* (de la serializare în fisiere plate sau XML) pâna la baze de date (relationale, relational-obiectuale sau orientate obiect) si în functie de *personalizarea sau externalizarea serviciilor suport* (tehnici de mapare relational/obiectual proprii sau preluate de la un furnizor specializat si codificarea proprie a accesului la date folosind direct API-ul si bibliotecile suport ale producatorului SGBD-ului sau preluarea de drivere sau adaptere furnizate de producatorul SGBD-ului).

Abordari în construirea unui strat de persistenta pentru aplicatiile de întreprindere

Realizarea unui (sub)strat de persistenta care sa acopere responsabilitatile nivelurilor 4-5 dintr-o arhitectura cu sase straturi trebuie sa tina seama de trei factori esentiali:

- *Închiderea (tranzitiva) a tuturor instantelor persistente* referentiate de (din) oricare instanta persistenta, care reprezinta graful complet format din toate referintele dintre obiectele persistente, referinte rezultate din câmpurile declarate explicit persistente (anumite câmpuri ar putea fi marcate *tranziente* însemnând ca valorile si deci eventualele referinte nu vor fi mentinute permanent). Capacitatea de a simula faptul ca respectivul graf se regaseste în memorie la discretia aplicatiei (desi el consta dintr-o serie de referinte care se regasesc în mare parte înca în „depozitul" de date, doar un subset al acestora fiind reconstituit în memorie) este unul din factorii care influenteaza decisiv gradul de transparenta a suportului pentru persistenta vizavi de natura exacta de stocare a datelor.
- *Transparenta suportului de persistenta* care se refera la *portabilitatea* între diversele paradigme de stocare (relationala, obiectuala etc.) si diferitele produse.
- *Transparenta modelului obiectelor domeniului,* care se refera în principal la gradul de libertate al proiectantului/dezvoltatorului aplicatiei fata de natura exacta a suportului de stocare (baza de date) si fata de eventuala existenta în prealabil a o parte din aceste structuri (structuri mostenite). Aceasta transparenta are de fapt corespondenta în *independenta logica a datelor* rezultata din arhitectura pe trei niveluri ANSI/SPARC. Cu alte cuvinte este vorba de abstractizarea modelului afacerii si apoi de aplicarea persistentei pentru acest model, si nu de a porni mai întâi de la modelul *cu persistenta* rezultat din structurile de stocare a datelor. Tinând cont de *nepotrivirea de trasaturi*, existenta spre exemplu între modelul obiectual al aplicatiilor (caracterizat de ierarhii de mostenire, agregare, suprascriere si supraîncarcare) si modelul relational plat, aceasta caracteristica devine extrem de importanta, chiar daca este vorba de BDO[1] care si ele trebuie proiectate luând în considerare un model de structurare unificat din care apoi sa fie derivate cele ale aplicatiilor individuale. Pentru sistemele de aplicatii noi (fara inconvenientul aplicatiilor mostenite) se recomanda formarea initial a unor echipe mixte între dezvoltatorii de aplicatii si cei ai bazelor de date. Scopul consta nu numai în obtinerea unor aplicatii robuste si extensibile, ci tin si de performanta: în ciuda transparentei, amplitudinea diferentelor dintre modelul obiectelor aplicatiei si cel al structurilor de stocare (baza de date) este invers proportionala cu performantele de accesare/reconstituire a datelor si de executare a interogarilor/actualizarilor corespunzatoare în bazele de date efective.
- *Limbajul de comunicare (interogare)* cu furnizorul de date (sau sursa datelor) care influenteaza direct portabilitatea. Scopul unui API gen JDBC sau ADO este crearea unei platforme portabile de acces la bazele de date relationale, limbajul desemnat fiind bineînteles ultimul standard ANSI-SQL. Din pacate diferitele personalizari si extensii proprietare depinzând de diversii producatori de SGBD-uri au afectat serios îndeplinirea acestui deziderat. În cazul bazelor de date orientate obiect, desi acestea ofera din start fiecare câte un API pentru persistenta nativa a obiectelor, inexistenta unui standard general acceptat în aceasta privinta si chiar imaturitatea modelului ODMG pentru BDO (a carui proces de acceptare între furnizorii de SGBO este mult prea lent) compromit destul de mult ideea de portabilitate chiar pentru mediile de persistenta considerate „native".

☑ **Persistenta obiectelor.** O *definitie* corecta si concisa a persistentei poate fi considerata urmatoarea: „un *obiect persistent* reprezinta un obiect care continua sa existe si dupa încheierea timpului de executie al programului care manipuleaza acel obiect" [Budd2002, 579]. Prin urmare persistenta ar fi de fapt capacitatea de a salva starea obiectelor pe un suport permanent si de a le reconstitui în mod transparent în spatiul de execu-

[1] Baze date orientate obiect sau baze de date obiectuale sau baze de obiecte

tie al aplicatiilor.

Din motive practice mai mult decât teoretice, calea de urmat ar fi sa se încerce a se asigura persistenta obiectelor într-o structura sau mediu de stocare relational, problema cheie fiind ca acest lucru sa se faca într-o maniera la fel de transparenta ca si seriabilizarea obiectelor în fisiere obisnuite. Problema ar putea fi sintetizata în obtinerea unei *seriabilizari relationale* a obiectelor gestionate în medii obisnuite orientate-obiect.

☑ **Natura *incompatibilitatii obiectual – relational*.** Pentru a sintetiza doar trei caracteristici esentiale care determina incompatibilitatea existenta între mediile obiectuale de dezvoltare a aplicatiilor si structurile de stocare relationale putem sublinia:

1. *Accesul* la obiecte/date: paradigma obiectuala presupune „traversarea" obiectelor prin intermediul referintelor dintre acestea, iar paradigma relationala presupune reunirea structurilor separate (tabelelor) prin intermediul valorilor atributelor cu rol de chei straine [Ambler2000_2].
2. *Identitatea*: paradigma obiectuala considera identitatea obiectelor ca fiind independenta de valorile atributelor, paradigma relationala conteaza cel mai mult pe cheile primare (eventual compuse) construite pe baza valorii câmpurilor.
3. *Ierarhiile*: paradigma obiectuala considera ierarhiile (de mostenire sau de agregare) ca fiind o cale naturala de construire a modelelor, pe când paradigma relationala aplatizeaza ierarhiile rezultând o serie de structuri relationale intermediare care nu reflecta în mod direct obiecte din lumea reala, ci aspecte particulare ale legaturilor (de asociere multipla, de generalizare, de compunere etc.) dintre acestea.

☑ **Conceptii în privinta constituirii stratului de persistenta.** În practica, în functie de complexitatea aplicatiei (si oarecum si de abilitatea sau priceperea programatorului) exista mai multe *conceptii* în ce priveste asigurarea (sub)stratului de persistenta: codificarea directa, încapsularea dialogului SQL (prin intermediul bibliotecii suport pentru accesul direct la structurile de stocare) în clase specifice si construirea unui strat de persistenta robust tinând seama de aspecte mai complexe de natura tranzactiilor, maparii transparente a obiectelor cu structurile relationale, gestiunii colectiilor de obiecte etc.

Cu privire la aceste abordari se poate remarca ca nu reprezinta altceva decât o trecere de simplu la complex în trei pasi, cu un câstig la fiecare etapa în ceea ce priveste completitudinea solutiei si transparenta persistentei în programare.

1. Codificarea directa. În acest caz aveam de a face cu încapsularea directa în codul aplicatiei a codului sau apelurilor API ale bibliotecii suport pentru accesul la baza de date (gen JDBC). Pentru aplicatii de mica anvergura reprezinta o alternativa relativ ieftina din punctul de vedere al costurilor de dezvoltare si implementare. Însa este sacrificata „puritatea" orientata obiect a aplicatiei, codul acesteia fiind mixat cu elemente de natura relationala datorita codificarii directe în straturile superioare a dialogului SQL necesar obtinerii datelor persistente din structurile de stocare. De asemenea, cresterea complexitatii aplicatiei va determina ridicarea semnificativa a costurile întretinerii aplicatiei, mai ales în contextul modificarilor frecvente ale structurilor din bazele de date.

2. Încapsularea dialogului SQL. Aceasta abordare presupune construirea claselor corespunzatoare suportului pentru persistenta astfel încât sa încapsuleze în metode specifice codul necesar comunicarii (pe baza suportului gen JDBC) cu sursele de date. Tehnica cea mai cunoscuta consta în încapsularea detaliilor de acces si dialog SQL într-o clasa suport de sine statatoare (tehnica DAO – *Data Access Object* [Reed2002]). Tehnica DAO [Reed2002, 241] presupune pe scurt trei elemente:

- *Clasele din domeniul afacerii* care sunt clase obisnuite ce respecta sistemul de tipuri al mediului de programare si este proiectata dupa modelul componentelor, adica toate atributele sunt private, fiind accesibile însa din metodele de acces tipice get/set.
- *Interfata DAO*, care specifica metodele cu semnaturile lor pe care trebuie sa le implementeze toate clasele de acces. Aceasta interfata uniformizeaza aspectul (extern) al

claselor de acces ducând la fluidizarea codului si cresterea valorii proiectului.

- *Clasele de acces*, câte una pentru fiecare clasa din domeniul afacerii care reclama persistenta pentru obiectele sale. Aceste clase de acces încapsuleaza logica maparii cu structurile relationale, cel mai simplu în fraze SQL preconstruite parametrizate. Logica maparii este necesara pentru satisfacerea cerintelor de reconstituire a obiectelor din structurile relationale sau salvarii/actualizarii acestora.

Tehnica DAO are avantajul ca *separa* în mod evident programarea logicii claselor din domeniul afacerii de logica accesului la baza de date. Prin urmare în echipa de dezvoltare se poate face o separare a sarcinilor în *proiectantul/programatorul logicii aplicatiei legate strict de regulile de afacere* vizate si *proiectantul/programatorul logicii persistentei datelor*. Acesta este sensul interfetei DAO prezentata mai sus.

Critica acestei tehnici vine din neabordarea directa a unor probleme mai complexe dar care survin chiar si în aplicatiile obisnuite: tranzactiile si lucrul cu colectiile de obiecte. În acest sens, fie trebuie construite câteva mecanisme simple încapsulate în clasele DAO, plus câteva clase noi (gen *Transaction* si *ObjectCursor*) fie daca aplicatia este destul de complexa (de la câteva zeci de clase, medii de stocare multiple, componente distribuite) se poate recurge la una din abordarile prezentate în paragraful urmator (*Crearea unui strat robust pentru persistenta*).

3. Un strat de persistenta robust. Este o sarcina dificila si care va aduce într-adevar beneficii pentru aplicatii complexe în care separarea totala a codului orientat-obiect al aplicatiei de schema relationala a bazei de date constituie un factor esential în asigurarea mentenantei. Programatorul aplicatiei va lucra într-un context asemanator ca în varianta anterioara, însa sarcinile legate de asigurarea corecta a suportului pentru persistenta vor genera responsabilitati pentru o suma de clase ce lucreaza împreuna pentru: a asigura maparea transparenta a schemei OO într-o schema relationala (tinând cont de aspecte mai complexe cum ar fi ierarhiile de mostenire, sau legaturile asociative simple sau de agregare dintre obiecte), a asigura desfasurarea într-o maniera tranzactionala a operatiilor legate de persistenta obiectelor (interactiunea cu mecanismele complexe de tranzactii si concurenta ale SGBD-urilor), a centraliza eventual suportul pentru persistenta disponibil pe baza a mai multor surse (baze de date relationale, baze de date obiectual/relationale, fisiere independente) etc.

În literatura de specialitate exista cel putin doua abordari în proiectarea structurala a unui sistemul de persistenta robust pentru aplicatiile de întreprindere.

Astfel *Wolfgang Keller*[2] propune o tehnica de realizare a substratului de persistenta pornind de la o structura generica care aduce în modelul stratului de persistenta elemente avansate legate spre exemplu de suportul tranzactiilor si maparea dinamica obiectual/relationala. Foarte pe scurt responsabilitatile din modelul propus de Keller pot fi structurate astfel:

- reprezentarea modelului datelor adaptat la necesitatile aplicatiei, constituit pe baza *tipurilor* din domeniul aplicatiei (*Hierarchical View*);
- reconstituirea datelor (sub forma view-urilor ierarhice – *Hierarchical View*) din structurile relationale (*View Factory*);
- gestionarea datelor reconstituite din structurile fizice sub forma view-urilor ierarhice (*View Cache*);
- gestionarea contextului tranzactional (*Transaction Object*);
- comunicarea directa – initierea si gestionarea dialogului SQL – cu baza de date (*Physical Views*);
- maparea (dinamica) a structurilor de date prezentate sub forma view-urilor ierarhice cu structurile de mapare prezentate sub forma *Physical Views* (*Query Broker*)

Pe lânga viziunea lui Keller mai poate fi adusa în discutie si abordarea din [*Ambler*[3] 2000_1] despre proiectarea unui strat robust pentru persistenta ce poate fi privita ca o extensei a celei dintâi. Foarte pe scurt re-

[2] → SD&M – Software Design & Management Gmbh & Co (www.objectarchitects.de)

[3] → *Ronin International Inc*, white papers despre Object Oriented Development la www.ambysoft.com

sponsabilitatile din modelul propus de Keller pot fi structurate astfel:

- reprezentarea claselor din domeniul afacerii extinse pentru a capata caracteristici de persistenta (regasire, salvare stergere) se face prin derivarea clasei *PersistentObject*;
- colectiile de obiecte persistente rezultate în urma dialogului cu bazele de date se realizeaza prin clasele *Cursor*, care detin instante *PersistenceObject* ce pot fi parcurse iterativ;
- clasele *PersistentObject* colaboreaza pentru maparea cu structurile fizice de stocare (îndeosebi relationale, dar mecanismul poate fi extins si pentru altele, de exemplu baze de date orientate obiect – sau baze de obiecte) cu clasele *ClassMap* care au ca responsabilitate construirea frazelor SQL necesare (instante ale unei subclase *SqlStatement*). Crearea dinamica a frazelor SQL se face incluzând în aceasta colaborare o ierarhie de clase destul de complexa, *PersistenceCriteria*, care încapsuleaza comportamentul necesar pentru a regasi, actualiza sau sterge colectii de obiecte (*Cursoare* de obiecte), pe baza unor criterii generice predefinite;
- tranzactiile se realizeaza evitând salvarea obiectelor direct din clasele *PersistentObject* corespunzatoare, si efectuând aceste operatii apelând la metodele unei instante dintr-o clasa numita *PersistentTransaction*;
- responsabilitatea colaborarii direct cu structurile de stocare fizice (îndeosebi baze de date relationale) cade în sarcina instantelor claselor numite *PersistenceMechanism*, ele încapsuleaza complexele biblioteci de clase cum sunt ODBC, JDBC sau ADO.

Trasatura esentiala a acestei tehnici rezida în faptul ca programatorul aplicatiei se bucura de transparenta totala a persistentei, singurele clase de care trebuie sa aiba cunostinta sunt *PersistentObject*, *PersistentCriteria*, *PersistentTransaction* si *Cursor*.

Concluzii

Proiectarea *modelului pentru asigurarea persistentei* datelor cu scopul de a furniza suportul pentru serviciile de mapare si acces cunoaste mai multe abordari care au ca principal obiectiv *prezervarea modelului obiectual al aplicatiilor* fata de natura diferita a structurilor datelor din sursele de date eterogene (în special relational, dar si obiectual/relationale, obiectuale sau chiar fisiere de date). Diferitele implementari tehnologice ale acestor abordari trebuie însa evaluate cel putin din urmatoarele doua puncte de vedere: *transparenta* si *portabilitatea*.

În consecinta, solutia *proiectare conceptuala orientata obiect + model logic obiectual al aplicatiilor + substrat pentru persistenta robust în privinta eterogenitatii surselor de date + SGBD relational*, cea mai acceptata la nivelul dezvoltatorilor aplicatiilor si dezvoltatorilor bazelor de date, este strict dependenta de gradul de portabilitate fata de nivelul datelor.

Bibliografie

[Ahmed &Umrysh2002] Ahmed, Khawar Zaman si Umrysh, Carry E. Developing Enterprise Java Applications with J2EE and UML, Addison-Wesley, 2002

[Ambler2000_1] Ambler Scott W., The Design of a Robust Persistence Layer For Relational Databases, Ronin International–White Papers, Nov. 2000 http://www.ambysoft.com/persistenceLayer.pdf)

[Ambler2000_2] Ambler Scott W., Mapping Objects To Relational Databases, Ronin International – White Papers, Oct. 2000 (http://www.ambysoft.com/mappingObjects.pdf)

[Atkinson89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto, în Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pg. 223-40, Kyoto, Japan, Decembrie 1989

[Budd2002] Budd, Timothy An introduction to object-oriented programming Third Edition, Addison Wesley, Pearson Education, Inc., 2002

[Reed2002] Reed, Paul R. Developing applications with Java and UML, Addison-Wesley, 2002

[Szabo98] Szabo, George, Client/Server Basics, InformIT (www.informit.com), 14 octombrie 1998