Object Role Modeling & Visual Studio .NET

Lect. Gabriela VÎRLAN
Catedra de Contabilitate si Informatica Economica, Universitatea „Dunarea de Jos" Galati

Database modeling represents an important step in designing a software application. Software tools allow minimizing the designer's computer work on the database much easier. At the same time, the number of the organizations that want to develop their own business over the Internet is continuously growing. Consequently, due to the tendencies mentioned above, the appearance of last generation products has been noticed on the software application market. Moreover, to the already existent software tools have been added new facilities which allow a relatively short time in designing these applications due to the fact that a series of tools which allow more operations at the same time are incorporated in them. Out of the numerous developing tools, the Microsoft family ones draw a special attention on an ease learning, and on the facilities they provide in or, designing and implementing some complex software applications.
***Keywords**: ORM, business rules, reverse engineering constraints, fact types, objects, entities.*

Introducere

Instrumentele de dezvoltare joaca un rol fundamental în utilizarea efectiva a tehnologiei informatiei pentru a construi ceea ce se numeste în momentul de fata Sistem Nervos Digital (*Digital Nervous System* – include mai multe elemente care permit organizatiilor sa-si reorganizeze procesele de afaceri, si sa le extinda prin intermediul Internetului). Microsoft a început furnizarea unor astfel de instrumente înca de la începutul anului 1975. Astfel, în luna martie 1997 Microsoft introduce sistemul de dezvoltare Visual Studio™ 97, o suita întreaga a instrumentelor de dezvoltare vizuale, care apare în doua editii: *Professional* si *Enterprise* (completata cu facilitati de dezvoltare rapida a aplicatiilor de întreprindere). Înca de la început acest produs a fost adoptat repede de organizatii datorita facilitatilor pe care le oferea pentru dezvoltarea aplicatiilor.

În momentul de fata s-a ajuns la versiunea Microsoft® Visual® Studio.NET, care cuprinde trei editii: *Visual Studio.NET Enterprise Architect*, *Visual Studio.NET Enterprise Developer* si *Visual Studio.NET Professional*.

De ce Microsoft® Visio® for Enterprise Architects ?

În materialul de fata accentul este pus pe o componenta a acestui produs, care faciliteaza proiectarea si implementarea bazelor de date. Se stie ca pentru a prospera si a avea succes în afacerile sale, o organizatie trebuie sa dispuna de informatii de o calitate superioara si în cantitati suficiente. Astfel, viitorologul Alvin Toffler spunea printre altele ca *„Informatia si comunicatiile constituie tot mai mult o forta de productie, iar informatia acumulata, o parte a avutiei nationale"*. Fara existenta informatiilor o organizatie nu ar mai putea exista, iar pentru a prospera aceste informatii trebuie utilizate corect, pentru a se putea obtine rezultate care sa conduca la alegerea celei mai bune solutii pentru prosperitatea acesteia. Pentru ca acest lucru sa fie posibil de realizat, informatiile trebuiesc stocate, triate (filtrate) si analizate în functie de necesitatile aparute într-un anumit moment dat. Cea mai buna alegere pentru ca aceste informatii sa fie coerente si sa nu fie pierdute sau distruse, în mod intentionat sau nu, o reprezinta stocarea acestora în baze de date. Se stie ca baza de date constituie componenta fundamentala a sistemului informational al unei organizatii, iar dezvoltarea si utilizarea sa trebuie privite din perspectiva cerintelor mai largi ale acesteia.

Din aceasta perspectiva este foarte importanta alegerea instrumentului de lucru utilizat pentru proiectarea bazei de date, care repre-

zinta procesul de realizare a unui proiect pentru o baza de date, care va trata toate operatiile si obiectivele organizatiei.

Produsului Microsoft® Visio® for Enterprise Architects (VEA) este inclus în Microsoft® Visual Studio® Enterprise Architect Edition a produsului Microsoft® Visual Studio® .NET. Acest produs include solutii de modelare a bazelor de date pentru definirea schemei conceptuale, logice si fizice a bazei de date, posibilitatea utilizarii *reverse engineering*, dar si a *forward engineering*.

Totodata, acest produs permite si modelarea sistemelor utilizând limbajul UML, precum si utilizarea tehnologiei XML, pentru implementarea aplicatiei într-un intranet, sau extranet.

Prin produsul Microsoft® Visio® for Enterprise Architects proiectarea unei baze de date se poate efectua prin cele doua metode: pornind de la zero (elaborând pe rând cele trei modele: conceptuala, logica si fizica si facându-se abstractie de ceea ce exista în organizatie), sau pornind de la o baza de date existenta (utilizând reverse engineering, fie ca se doreste optimizarea acesteia, fie ca se doreste a se avea un punct de pornire pentru noua baza de date).

Ce reprezinta ORM?

Pentru a proiecta o baza de date este necesara construirea unui model formal al domeniului aplicatiei. Object Role Modeling (ORM) simplifica procesul de proiectare prin utilizarea unui limbaj natural, precum si a unor diagrame care pot fi populate cu exemple. Prin exprimarea modelului în termeni precum roluri si obiecte, ORM furnizeaza o modalitate conceptuala, usor de înteles si folosita pentru modelarea datelor. Modelul conceptual rezultat este independent de platforma pe care se va implementa baza de date.

ORM reprezinta o descriere generala a bazei de date, utilizând simboluri intuitive (pentru obiecte si atribute) si un limbaj natural pe care atât utilizatorii cât si proiectantii bazei de date le poate întelege si care verbalizeaza (exprima) relatiile dintre acestea, utilizând o gama larga de constrângeri care surprinde si forteaza regulile de business.

Versiunile timpurii ale ORM-ului au fost dezvoltate în Europa la mijlocul anilor '70 (de exemplu, modelarea relatiilor binare, Natural Language Information Analysis Method – NIAM, Natural Language Modeling – NLM, Format Object-Role Modeling Language – FORML). Dintre caracteristicile mai importante ale metodologiei ORM amintim:

- este *usor de înteles* – exprima fapte si reguli în limba engleza si/sau alte grafice intuitive;
- este *fiabil* – valideaza regulile folosind limba engleza si mostre de date;
- este *expresiv* – surprinde în mod grafic multe reguli de business si mai complexe;
- este *stabil* – minimizeaza impactul modificarilor asupra modelului si interogarilor.

O diagrama ORM poate fi folosita în loc de sau înaintea unui alt model de proiectare a bazei de date si permite:

- proiectarea conceptuala a bazei de date folosind fapte si exemple descrise într-un limbaj natural;
- construirea automata a modelelor logice si fizice a bazei de date pe baza modelului sursa ORM;
- modelul bazei de date este creat într-un limbaj care poate fi înteles de utilizatorii non-tehnici.

Pentru generarea unui model sursa ORM proiectantul are la dispozitie mai multe posibilitati:

- crearea unui model nou pornind de la zero;
- utilizarea unui model drept punct de plecare utilizând procedeul reverse engineering;
- importul si redefinirea unui model existent.

Exprimarea regulilor de business

ORM reprezinta o aplicatie ca un set de obiecte care joaca roluri (parti componente ale unei relatii). Din acest motiv ORM este numit uneori modelare bazata pe fapte. Principalele elementele de baza constructive utilizate în ORM sunt: tipuri obiect si relatiile. Tipurile obiect care pot fi utilizate sunt: entitate (*entity*), valoare (*value*), obiect extern (*external*).

Tipurile valoare si entitate sunt numite si tipuri obiect lexicale, respectiv nelexicale. De

asemenea, în ORM, **relatiile** dintre tipurile obiect entitate (numite si *fapte*), si relatiile dintre tipurile obiect entitate si tipurile obiect valoare (numite si *referinte*) sunt reprezentate prin predicate care contin unul sau mai multe roluri. Deci, tipurile de propozitii care pot fi utilizate sunt: tipuri de fapte (*fact types*) si tipuri de referinta (*reference types*).

Tipurile de fapte sunt introduse prin intermediul unui editor de fapte (Fact Editor), care prezinta si alte functii precum: introducerea tipurilor obiect, a unor exemple care sa permita validarea faptelor, a constrângerilor care sunt aplicate faptului respectiv etc.

Dupa editarea tuturor tipurilor de fapte acestea sunt aduse în zona pentru desenarea diagramei, care permite vizualizarea partiala sau integrala a modelului ORM. Astfel, tipurile obiect sunt reprezentate prin figuri geometrice ovale. Tipurile obiect sunt conectate între ele prin intermediul unor predicate, reprezentate ca o secventa de casete. Fiecare caseta corespunde unui rol din relatia existenta (figura 1).

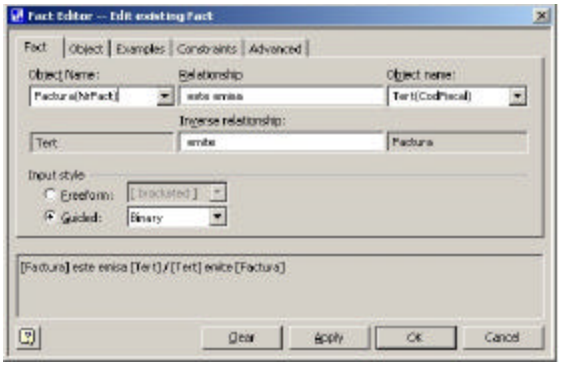

Fig.1.

Faptul introdus este exprimat de propozitia „Factura este emisa/emite Tert”, în care *tipurile obiecte* sunt Factura si Tert, iar relatiile sunt exprimate prin predicatul “este emisa de/emite”. În acest caz predicatul are doua roluri. De exemplu, tipul de fapt „Factura este emisa Tert” are un rol jucat de Factura (este emisa) si un rol jucat de Tert (emite). ORM permite utilizarea unuia sau mai multor roluri în cadrul unui predicat. NrFact si CodFiscal reprezinta referintele pentru tipurile obiect entitate.

Utilizarea constrângerilor

Modelul poate include regulile de afaceri, constrângeri sau roluri derivate. Constrângerile provin din raspunsurile la întrebarile puse într-un limbaj natural. În functie de numarul de predicate care intra în alcatuirea constrângerii, acestea se clasifica în doua tipuri:

a. constrângeri interne, caz în care constrângerea este aplicata asupra unui singur predicat;

b. constrângeri externe, caz în care constrângerea este aplicata la doua sau mai multe predicate.

Într-un model ORM pot fi utilizate urmatoarele tipuri de constrângeri:

1. Constrângeri de unicitate: interne, externe, primare.
2. Constrângeri obligatorii: simple, disjunctive.
3. Constrângeri circulare: aciclice, asimetrice, antisimetrice, intranzitive, nereflexive, simetrice.
4. Constrângeri comparare de multimi: submultimi, de egalitate, de excludere.
5. Alte tipuri de constrângeri: de frecventa, index, de valoare.

De exemplu, în figura 2, sageata arata existenta unei constrângeri de unicitate, deci fiecare obiect îndeplineste acest rol numai o singura data (Factura este emisa de cel mult un Tert).

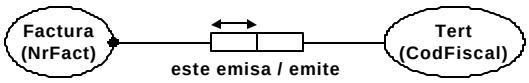

Fig.2.

Valorile posibile pe care le poate lua un tip obiect valoare pot fi indicate prin utilizarea constrângerilor de valoare. Aceasta constrângere se refera la domeniul de valori sau la valorile pe care tipul obiect valoare le poate lua.

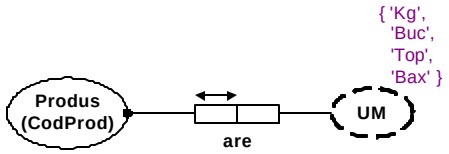

Fig.3.

Astfel, în figura 3 notatia {‘Buc’ ‘Kg’, ‘Top’, ‘Bax’} restrictioneaza valorile pe care le poate lua tipul obiect valoare UM. Pentru toate tipurile de constrângeri exista simboluri intuitive. Astfel, dupa definirea tuturor tipurilor

de fapte se va obtine modelul sursa ORM conceptual (figura 4).

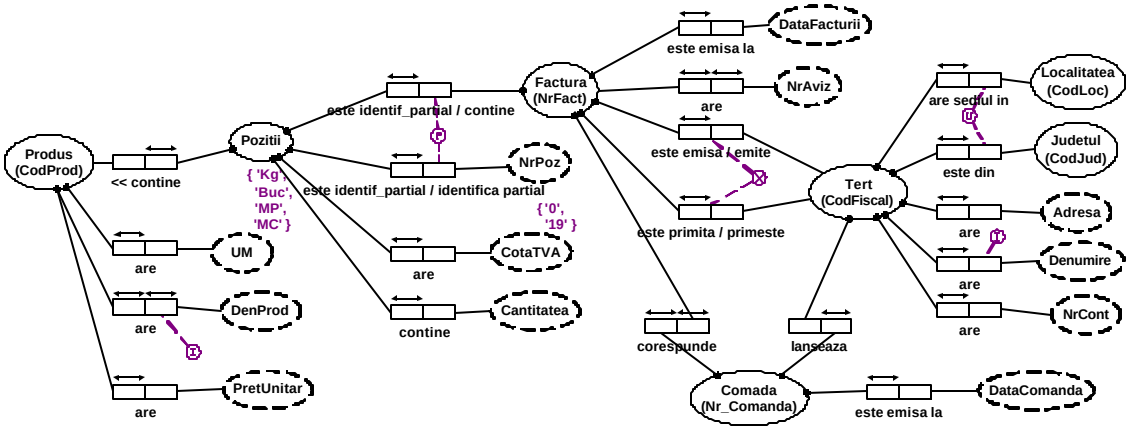

Fig.4.

Transpunerea unui model ORM într-un model logic al bazei de date

Transpunerea unui model sursa ORM într-un model logic al bazei de date presupune adaugarea modelului ORM la proiectul bazei de date ce se doreste a fi implementata si apoi construirea acestuia. Pentru obtinerea modelului logic al bazei de date proiectantul poate utiliza un model logic deja existent sau poate crea unul nou. În ambele situatii, dupa deschiderea sau dupa crearea modelului logic, i se va adauga modelul sursa ORM. Înainte de generarea tabelelor si a relatiilor dintre acestea, instrumentul va efectua o verificare automata a modelului sursa ORM. În situatia în care acesta contine erori (predicate în conflict, relatii subclasa superclasa lipsa etc.) Microsoft® Visio® for Enterprise Architects nu va genera modelul logic al bazei de date, indicând erorile care nu permit generarea modelului. Astfel, modelul logic rezultat pe baza modelului sursa ORM va arata ca în figura 5. Reprezentarea din cadrul modelului logic (ca si în cel fizic, de altfel) este o reprezentare ERD, folosind implicit notatia relationala.

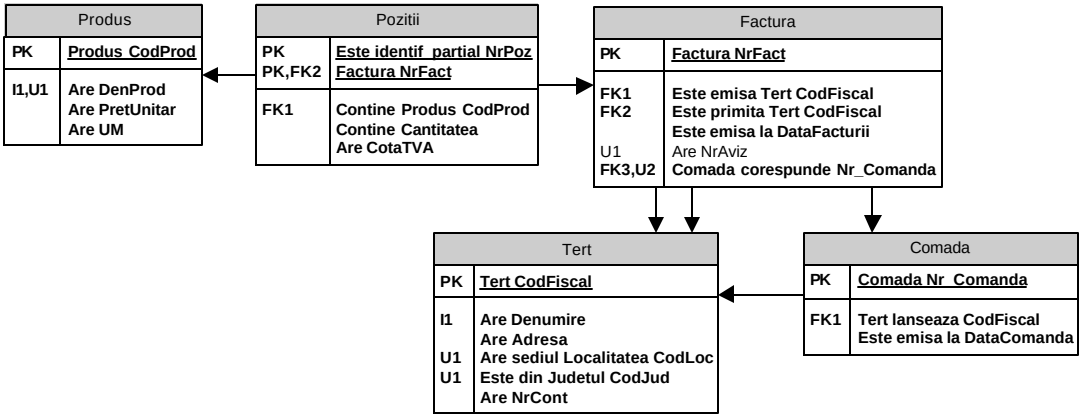

Fig.5.

Fiecare tabela are propriul sau nume afisat în antet (de exemplu, Factura), dupa care urmeaza lista coloanelor. Cheile primare sunt afisate subliniat si marcate cu PK (*Primary Keys*) si apar la începutul listei coloanelor. Coloanele obligatorii sunt afisate boldat. Coloanele care sunt chei straine sunt marcate cu FK*n* (*Foreingn Keys*), unde *n* reprezinta numarul cheii straine din tabela. Abrevierea U*n* reprezinta o constrângere de unicitate, unde *n* reprezinta numarul constrângerii de unicitate din cadrul tabelei.

În acest exemplu, numele tabelelor si ale coloanelor sunt cele generate în mod implicit din modelul sursa ORM (conceptual). În practica se pot atribui alte denumiri pentru

tabele sau pentru coloane, si de asemenea se pot modifica tipurile de date implicite care au fost desemnate de sistem pentru fiecare coloana a tabelei.

Referitor la tipurile de date este recomandat ca acestea sa fie setate atunci când se elaboreaza modelul ORM, unde tipurile de obiecte corespund cu domeniile conceptuale. Astfel, tipurile de date corecte sunt apoi în mod automat propagate catre toate atributele care au la baza aceste domenii de valori (figura 6).

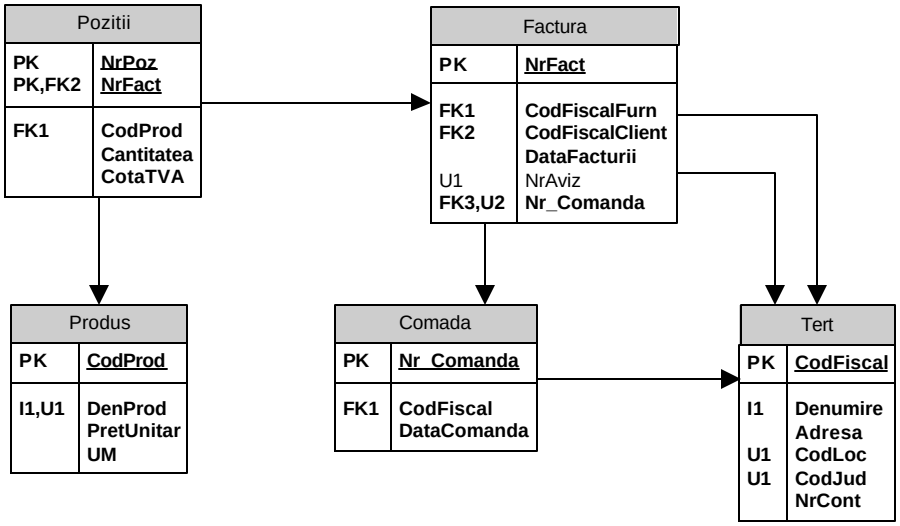

Fig.6.

În acest moment tipurile de date cu care se lucreaza sunt datele portabile, independente de platforma. La acest nivel, modelul contine atributele grupate pe entitati, relatiile definite între aceste entitati, indexuri si cheile definite la nivelul fiecarei entitati etc. Se pot efectua modificari în cadrul acestui model, pâna când se va considera ca s-a atins varianta optima.

Comparând cu modelarea logica a bazei de date, ORM descrie faptele afacerii în propozitii simple, în timp ce schema SGBDR utilizeaza tabele cu atribute. Transpunerea constrângerilor în modelul logic al bazei de date este executata automat. Astfel, constrângerile de unicitate sunt transpuse în chei primare (PK) sau constrângeri de unicitate (U1). Anumite constrângeri (de valoare, obligatorie disjunctiva sau exclusiva) nu au o reprezentare în modelul logic al bazei de date, dar ele sunt reprezentate în codul DDL atunci când este generata baza de date fizica.

Generarea schemei fizice a bazei de date

Urmatorul pas îl reprezinta generarea schemei interne pentru sistemul de gestiune a bazei de date ales. Se poate alege optiunea de generare a unui script DDL. Este optiunea cea mai buna de ales, deoarece script-ul DDL generat poate fi utilizat mai departe în interiorul SGBD-ului ales. Ceilalti pasi care trebuiesc urmati se refera la: alegerea driver-ului (de exemplu, Microsoft® Access, Microsoft® SQL Server 2000™ etc.); introducerea unui nume pentru baza de date (de exemplu, Facturi), acceptarea setarilor implicite si apoi generarea script-ului DDL, care poate fi salvat ca un fisier text, xml, html etc.

În exemplu utilizat în material, baza de date a fost creata în SQL Server 2000. Diagrama creata în urma generarii bazei de date este ilustrata în figura 7.

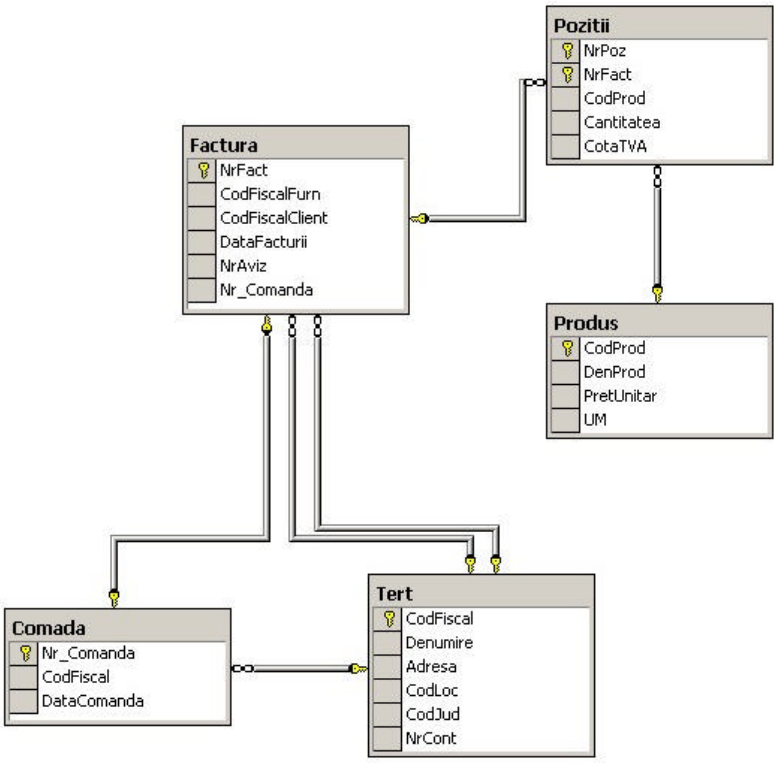

Fig.7.

Concluzii

Din materialul de fata reies facilitatile pe care le ofera acest instrument: generarea schemei bazei de date, (direct sau printr-un script DDL), verificarea erorilor modelului, sincronizarea modelului cu baza de date, colaborarea si lucrul în echipa, proiectarea bazei de date din perspectiva regulilor de business, generarea diagramelor UML, generarea unui fisier XML dintr-un model ORM.

Bibliografie

📖 Catalin Raicu, *Modelarea bazelor de date cu VISIO 2000*, Microsoft® TechNet Start, Anul 2, Nr.4, Octombrie 2001, pag.26-28

📖 John Sharp, *Validating Information Models*, Journal of Conceptual Modeling, March 2002, http://www.inconcept.com

📖 Terry Halpin, *An ORM metamodel of Information Engineering*, Journal of Conceptual Modeling, February 2001, http://www.inconcept.com/jcm

📖 Terry Halpin, *Microsoft's New Database Modeling Tool*, Journal of Conceptual Modeling, June 2001, http://www.inconcept.com

📖 Terry Halpin, *Object Role Modeling An Overview*, Microsoft Corporation, November 2001, http://msdn.microsoft.com

📖 Terry Halpin, *Visio-Based Database Modeling in Visual Studio.NET Enterprise Architect*, Microsoft Corporation, http://www.msdn.microsoft.com

📖 Th. Connoly, C. Begg, A. Strachan, *Baze de date. Proiectare, implementare, gestionare*, Editura Teora, Bucuresti, 2001, ISBN 973-20-0601-3