VYUŽITÍ NÁVRHOVÉHO VZORU ADAPTÉR PRO ODSTÍNĚNÍ ROZDÍLŮ V API OBJEKTOVÝCH DATABÁZÍ

Vítězslav Novák¹

¹ Vysoká škola báňská-Technická univerzita Ostrava, Ekonomická fakulta, Sokolská třída 33, 701 21 Ostrava
Email:vitezslav.novak@vsb.cz

Abstract: Actual ERP systems use for data storage mainly relational databases, but in specific situations, particularly for modeling complex relationships can also use the object databases. These information systems are usually developed using object-oriented programming languages like C++, Java, C #, and it would seem natural that for the data storage they also use the object-oriented databases instead of the relational databases, thus dispensing with the need for mapping objects into relations. But this is not true especially because of two reasons. The first reason is historical and the resulting enormous widespread of the relational databases, which were here before the object-oriented databases. The second reason for minority use of the object-oriented databases is the minimum compliance of standards by the object DBMS (Database Management System) producers, such as SQL standard in relational databases, resulting in differences in the API (Application Programming Interface) of object-oriented DBMS and the impossibility to change the object-oriented DBMS in an application without having to rewrite the code of an entire application which is built on that object-oriented database. The following paper attempts to resolve this issue.

Keywords: adapter, application programming interface, design pattern, object database, query language.

JEL classification: L86

Doručeno redakci: 7.2.2011; Recenzováno: 14.12.2012; 1.3.2013; Schváleno k publikování: 16.4.2013

Úvod

Objektové databáze v současné době zajisté nejsou dominantním typem používaných databází. Přesto jsou stále vyvíjeny a používány. Stačí se podívat na webové stránky firem vyvíjejících objektové SŘBD. Namátkou lze zmínit objektové SŘBD GemStone, Caché, Objectivity/DB, Versant Object Database a další. Z implementací objektových databází lze zjistit, že k jejich nasazování dochází v tom okamžiku, kdy je modelovaná skutečnost natolik složitá, že vývoj aplikací na relační úrovni se stává neúměrně drahý, pomalý nebo složitý.

Vedle komerčních objektových SŘBD, které jsou obvykle velmi drahé, je ale vyvíjeno také několik objektových SŘBD jako open source projekty, mimo jiné např. Ozone, ObjectDB, db4o a další. Vývoj objektových SŘBD začal ruku v ruce s vývojem objektově orientovaných programovacích jazyků zhruba v polovině 80. let. V té době už ale byly relační SŘBD hluboce zakořeněné v podnikové praxi a jejich pozice je dodnes neotřesitelná. Proto se objektové databáze používají jen minimálně, více méně pouze v tom okamžiku, kdy je modelovaná skutečnost natolik složitá, že vývoj aplikací na relační úrovni se stává neúměrně drahý, pomalý nebo složitý. Důvodem tohoto stavu je ale nejen historické prvenství relačních SŘBD, ale zejména nedodržování standardů ze strany výrobců objektových SŘBD. Relační SŘBD používají standard SQL (Structured Query Language), což aplikacím umožňuje jak jednotný přístup k datům relačních databází, ať už se jedná o databáze Oracle, SQL Server, DB2, MySQL nebo jiné, tak relativně snadnou přenositelnost na konkurenční SŘBD.

Zde je nutno poznamenat, že tradiční výrobci relačních SŘBD jako IBM, Oracle nebo Microsoft také zareagovali na objektově orientované trendy v oblasti vývoje aplikací a pokusili se do svých relačních SŘBD přidat objektové rysy, čímž vznikla nová kategorie tzv.

objektově-relačních SŘBD. Objektově-relační SŘDB využívají datový model tak, že přidávají objektové rysy do tabulek. Všechny trvalé informace jsou stále v tabulkách, ale některé položky mohou mít bohatší datovou strukturu – tzv. abstraktní datové typy. Abstraktní datový typ je datový typ, který vznikne zkombinováním základních datových typů.

Objektově-relační prvky jsou postupně zahrnovány do standardu jazyka SQL a to od standardu SQL3 (SQL-99). Objektově-relační SŘDB jsou tedy nadmnožinou relačních SŘBD a pokud nevyužijeme žádná objektové rozšíření, jsou ekvivalentní standardu SQL2. Podle Connolly (2010) hlavní výhoda objektově-relačních SŘBD spočívá v tom, že umožňuje využít významnou část znalostí a zkušeností získaných při vývoji relačních databází. Hlavní nevýhodu naopak vidí v růstu komplexnosti takovýchto SŘBD, neboť jednoduchost čistě relačního přístupu je jednou z jeho hlavních výhod.

1 Standardizace objektově orientovaných databází

Co se týče standardizace objektových SŘBD, v roce 1993 skupina ODMG (Object Database Management Group) vydala standard ODMG93, při jehož tvorbě se inspirovala standardy vydanými OMG (Object Management Group). Zatím poslední verze standardu je ODMG 3.0 z ledna 2000. Hlavním cílem standardu ODMG je poskytnout společné rozhraní, pomocí kterého by bylo možné vyvíjet aplikace, které by byly přenositelné mezi různými objektovými databázovými platformami. Pro plnou přenositelnost je nutné standardizovat datový model, vazby na programovací jazyky a jazyky pro manipulaci a definici dat, přičemž jazykem pro manipulaci s daty (DML) je samotný programovací jazyk (např. Java). Standardy ODMG podle Cattell (2000) sestávají zejména z těchto částí:

- Jazyk pro definici objektů ODL (Object Definition Language) – slouží k definici objektů, jejich atributů a metod.
- Dotazovací jazyk OQL (Object Query Language) – objektový dotazovací jazyk, který vychází z SQL92 a rozšiřuje jeho možnosti o práci s objekty.
- Jazyk pro manipulaci s objekty OML (Object Manipulation Language).
- Definice přiřazení objektů uložených v databázi do programovacích jazyků C++, Smalltalk a Java.

Bohužel tento standard není výrobci objektových SŘBD respektován, což má za následek rozdílné API pro přístup k objektovým databázím a z toho vyplývající nemožnost změnit objektový SŘBD bez nutnosti přepsat celou aplikaci, která je na této databázi postavena. Následující tabulka demonstruje rozdílnost API několika vybraných objektových SŘBD na základních třídách a metodách v programovacím jazyce Java umožňujících vybírání objektů, jejich přidávání, změny a odstraňování, tedy objektové alternativy základních příkazů jazyka SQL – Select, Insert, Update, Delete:

Tabulka 1: Základní třídy a jejich metody vybraných objektových SŘBD

	ObjectDB	db4o	Caché
Operace	Přidávání, změny a odstraňování objektů		
Třída	javax.jdo.PersistenceManager	com.db4o.ObjectContainer	com.intersys.objects.Database
Metody	void makePersistent(Object object)	void set(Object object)	void createObjects(String cacheClassName, Collection objects)
	void makePersistent(Object object)	void set(Object object)	void saveObjects(String cacheClassName, Collection objects)
	void deletePersistent(Object object)	void delete(Object object)	void deleteObjects(String cacheClassName, Collection objects)
	javax.jdo.Query newQuery(Class c, String filter)	com.db4o.query.Query query()	<i>tento mezikrok je vynechán</i>
Operace	Dotazování		
Dotaz. jazyk	JDOQL	Query by Example, Native Queries, SODA	SQL
Třída	javax.jdo.Query	com.db4o.query.Query	<i>tento mezikrok je vynechán</i>
Metoda	java.util.Collection execute()	com.db4o.ObjectSet execute()	java.util.Iterator openByQuery(String sql)

Zdroj: vlastní zpracování

V této tabulce lze názorně vidět, že každý objektový SŘBD používá zcela odlišné API i objektový dotazovací jazyk, i když principy práce jsou mnohdy velmi podobné. Proto pokud je již některý objektový SŘBD použit, pak řešení postavené na tomto SŘBD bývá vysoce specializované pro tento SŘBD a tudíž velmi těžce přenositelné na jiný. Pokud má být zachována možnost přenositelnosti klientské aplikace na jiný objektový SŘBD, pak z této klientské aplikace není možné volat přímo metody objektového SŘBD, protože to by znamenalo při změně objektového SŘBD přepsat celou klientskou aplikaci. Aby byla klientská aplikace přenositelná na jiný objektový SŘBD, je nutno mezi aplikací a databází vložit určitou softwarovou mezivrstvu, která odstíní rozdíly mezi aplikací a databází, která databázi adaptuje na klienta. A tady se dostáváme k návrhovému vzoru Adaptér.

2 Návrhový vzor Adaptér

Návrhový vzor Adaptér byl poprvé publikován v Gamma (1995) mezi 23 základními návrhovými vzory. Podle autorů této publikace jsou návrhové vzory popisy komunikujících objektů a tříd, které jsou upraveny k řešení obecného návrhového problému v určitém kontextu.

Návrhový vzor Adaptér podle Gamma (1995) převádí rozhraní jedné třídy na jiné rozhraní, které očekává klient. Adaptér umožňuje spolupráci tříd, které by spolu nefungovaly z důvodu nekompatibilního rozhraní. Vzorek Adaptér lze použít v těchto případech:

- Máme použít již existující třídu, ale její rozhraní se neshoduje s tím, které potřebujeme.
- Chceme vytvořit třídu k znovupoužití, která spolupracuje s nepříbuznými nebo neznámými třídami, tj. takovými, jež nemusí mít nutně kompatibilní rozhraní.
- Musíme použít několik existujících podtříd, ale je nepraktické přizpůsobit jejich rozhraní vytvořením podtřídy pro každou z nich. Objektový adaptér umí přizpůsobit rozhraní své rodičovské třídy.

Návrhový vzor Adaptér lze použít jako třídní adaptér, který využívá vícenásobnou dědičnost k přizpůsobení dvou rozhraní, nebo jako objektový adaptér, který spoléhá na objektovou skladbu. Tento lze schematicky vyjádřit následujícím diagramem tříd:

Obrázek 1: Diagram tříd návrhového vzoru Adaptér

Zdroj: Gamma (1995)

Jak lze z výše uvedeného obrázku vidět, návrhový vzor Adaptér využívá následující princip. Klient (*Client*) vyžaduje pro svou funkčnost určité cílové rozhraní (*ITarget*) dané souborem několika operací (*operation()*). Adaptovaná třída (*Adaptee*) ale disponuje jiným souborem operací (*specificOperation()*), než vyžaduje klient. Pro vyřešení tohoto problému je vytvořen adaptér (*Adapter*) jako implementace cílového rozhraní (*ITarget*). Tento adaptér jako člena obsahuje odkaz na instanci adaptované třídy a zároveň implementuje všechny operace cílového rozhraní (*ITarget*). Z těchto operací pak pouze volá specifické operace adaptované třídy (*Adaptee*).

V případě, že se změní adaptovaná třída, je pouze nutné změnit adaptér, jehož operace teď budou volat jiné specifické operace adaptované třídy, ale podstatné je, že klient zůstává naprosto beze změn. V tomto článku bude návrhový vzor Adaptér použit pro odstínění rozdílů v API jednotlivých objektových SŘBD.

3 Odstínění rozdílů v API jednotlivých objektových SŘBD

V této kapitole se pokusím zjednodušeně demonstrovat, jakým způsobem je možné výše zmíněný návrhový vzor Adaptér použít pro odstínění rozdílů v API jednotlivých objektových SŘBD. Pro příklad jsem zvolil objektové SŘBD ObjectDB a db4o. Samozřejmě při použití jakéhokoli jiného objektového SŘBD by se postupovalo stejně.

Dejme tomu, že klientská aplikace vyžaduje pouze následující tři základní databázové operace pro ukládání, změnu a odstraňování objektů u objektové databáze:

- void insert(Object o),
- void update(Object o),
- void delete(Object o).

Z tabulky základních tříd a jejich metody vybraných objektových SŘBD uvedené výše je vidět, že oba objektové SŘBD mají pro tyto operace své zcela odlišné specifické operace. Z tohoto důvodu je proto nutné vytvořit adaptér, který bude operace vyžadované klientem překládat na operace, kterým bude rozumět konkrétní objektový SŘBD, viz následující obrázek použití objektového SŘBD ObjectDB:

Obrázek 2: Použití objektového SŘBD ObjectDB

Zdroj: vlastní zpracování

Pokud bychom se ale rozhodli v aplikaci vyměnit objektový SŘBD z výše zmíněného ObjectDB na db4o, stačilo by pouze změnit volání operací adaptované databáze v operacích adaptéru. Klient by zůstal zcela beze změny, viz následující obrázek:

Obrázek 3: Použití objektového SŘBD db4o

Zdroj: vlastní zpracování

Protože se databázové operace jednotlivých objektových SŘBD principiálně neliší, nýbrž se liší pouze v názvech jednotlivých operací, jejich parametrech nebo návratových typech, mělo by být vždy možné tyto operace adaptovat pomocí adaptéru. To už ale bohužel zcela neplatí pro dotazovací jazyky objektových SŘBD.

4 Problém dotazovacích jazyků

Ve výše zmíněné tabulce základních tříd a jejich metod vybraných objektových SŘBD jsou uvedeny názvy dotazovacích jazyků těchto SŘBD. Tyto dotazovací jazyky se ale liší velmi podstatně. Jako příklad si můžeme uvést již použité objektové SŘBD ObjectDB a db4o, viz následující výpis kódu v programovacím jazyce Java:

```
//ObjectDB - dotazovací jazyk JDOQL
Query query = pm.newQuery(BeznyUcet.class, "this.nazevUctu == \"Novák\"");
Collection col = (Collection) query.execute();

//db4o - dotazovací jazyk SODA
Query query = db.query();
query.constrain(BeznyUcet.class);
query.descend("nazevUctu").constrain("Novák");
ObjectSet ucty = query.execute();
```

Jak lze vidět ve výše uvedeném výpisu, objektový SŘBD ObjectDB využívá dotazovací jazyk JDOQL, který pro zadávání kritérií výběru používá výrazy podobné výrazům používaným v klauzuli WHERE příkazu SELECT jazyka SQL. Naproti tomu objektový SŘBD db4o využívá dokonce tři různých dotazovacích jazyků, kde v ukázce je použit dotazovací jazyk SODA, který pro zadávání kritérií výběru zcela využívá programovací jazyk Java.

Přes tyto zásadní rozdíly v dotazovacích jazycích lze ale i zde vystopovat alespoň nepatrný průnik v použitých principech dotazování. Každý dotazovací jazyk totiž obsahuje:

- název třídy, jejichž instance budou z databáze vybírány,
- název atributu, na kterém bude výběr prováděn,
- omezení pro filtrovací atribut,
- dotaz vrací nějaký typ kolekce.

Z tohoto důvodu lze předpokládat, že by cílové rozhraní klienta mohlo obsahovat také operace pro výběr objektů z objektové databáze s výše zmíněnými parametry a tyto operace by pak bylo možno adaptovat v adaptéru na libovolný objektový SŘBD. Metody umožňující dotazovací operace by mohly mít např. následující signatury:

- `java.util.List execute(java.lang.String type)` - výběr vše instancí daného typu.
- `java.util.List execute(java.lang.String type, java.lang.String fieldName, java.lang.String constrain)` - výběr všech instancí daného typu vyhovující zadanému kritériu pro daný atribut.

Toto řešení sice způsobí ne plné využití možností dotazovacích jazyků jednotlivých objektových SŘBD, na druhou stranu ale klientská aplikace zůstane přenositelná na jiný objektový SŘBD beze změny kódu klientské aplikace.

Závěr

Článek se snažil poukázat na problémy související s použitím objektových databází při tvorbě aplikací. Hlavním problémem se v tomto směru jeví nedodržování standardů a z toho vyplývající rozdílnost jak v API jednotlivých objektových SŘBD, tak v používaných objektových dotazovacích jazycích. Tato rozdílnost způsobí, že v případě změny objektového SŘBD za jiný je nutno přeprogramovat celou klientskou aplikaci. Aby to nebylo nutné, je možno mezi klientskou aplikací a objektovou databází vložit speciální softwarovou

mezivrstvou postavenou na principu návrhového vzoru Adaptér, která tyto rozdíly odstíní. Je možné, že v tomto případě nebude možné využívat celou škálu schopností daného objektového SŘBD, přenositelnost klientské aplikace na jiný objektový SŘBD však zůstane zachována. Z dostupných zdrojů nebylo zjištěno, zda-li se problém různých API objektových SŘBD snažil někdo jiný řešit jiným způsobem.

Literatura

- [1] CATTELL, R., D. BARRY a M. BERLER, 2000. *The Object Data Standard: ODMG 3.0*. San Francisco: Morgan Kaufmann. ISBN 15-586-0647-4.
- [2] CONNOLLY, Thomas M. a Carolyn E. BEGG, 2010. *Database systems: a practical approach to design, implementation, and management*. 5. vyd. London: Addison-Wesley. ISBN 03-215-2306-7.
- [3] db4objects. *db4o: Java & .NET Object Database*. [online]. Dostupné na: <http://www.db4o.com>
- [4] GAMMA, E., R. HELM, R. JOHNSON a J. VLISSIDES, 1995. *Design patterns : elements of reusable object-oriented software*. 1. vyd. Reading: Addison-Wesley, 395. ISBN 0-201-63361-2
- [5] Object Database Management Systems. *ODBMS.ORG: The Resource Portal for Education and Research*. [online]. Dostupné na: <http://www.odbms.org/odmg/>
- [6] ObjectDB. *ObjectDB – Fast Object Database for Java with JPA/JDO Support*. [online]. Dostupné na: <http://www.objectdb.com>