# Vlastnosti objektově orientovaného datového modelu.

Thursday, May 30, 2013 8:19 AM

# The Object-Oriented Data Model

1. A data model is a logic organization of the real world objects (entities), constraints on them, and the relationships among objects. A DB language is a concrete syntax for a data model. A DB system implements a data model.
2. A core object-oriented data model consists of the following basic object-oriented concepts:
   (1) **object and object identifier**: Any real world entity is uniformly modeled as an object (associated with a unique id: used to pinpoint an object to retrieve).
   (2) **attributes and methods**: every object has a state (the set of values for the attributes of the object) and a behavior (the set of methods - program code - which operate on the state of the object). The state and behavior encapsulated in an object are accessed or invoked from outside the object only through explicit message passing.
   [ An attribute is an instance variable, whose domain may be any class: user-defined or primitive. A class composition hierarchy (aggregation relationship) is orthogonal to the concept of a class hierarchy. The link in a class composition hierarchy may form cycles. ]
   (3) **class**: a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance-of relationship). A class is similar to an abstract data type. A class may also be primitive (no attributes), e.g., integer, string, Boolean.
   (4) **Class hierarchy and inheritance**: derive a new class (subclass) from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class and may have additional attributes and methods. single inheritance (class hierarchy) vs. multiple inheritance (class lattice).

From <http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter8/node3.html>

HDM, SDM a RDM = záznamově orientované modely. V 90. letech se začaly objevovat první objektově orientované SŘBD (OOSŘBD), které umožňují pracovat s datovou abstrakcí na úrovni objektů.

- Výhody
  - snadnější aktualizace dat
  - přímé vyjádření složitých objektů modelované reality v databázi (odpadají mezikroky převodu objektů do normalizovaných tabulek relační databáze. Stejně tak je zjednodušen i opačný krok načítání objektů z databáze do aplikace)
  - součástí uložených objektů je také jejich chování (metody atd.)
- Nevýhoda
  - Vývoj a návrh objektově orientovaných modelů v jejich univerzálnosti a komplexnosti je velmi složitý proces ⇒ menší uplatnění tohoto typu modelu v reálných aplikacích.

**Objektově orientované programování (OOP)**

Koncepce objektově orientovaných databází vychází z principů používaných v OOP - základem je objekt (prvek typu třída). Data se nazývají atributy, funkce metody (služby). Metoda je aktivována příchodem zprávy do jiného objektu.

Hlavní vlastnosti OOP: zapouzdření dat, dědičnost, polymorfismus

**Objektově orientované modelování dat**

Postup objektově orientovaného modelování dat lze stručně shrnout do následujících bodů:

- Vyhledají se objekty jako nositelé aktivit (např. metodou gramatické inspekce: podstatná jména představují objekty nebo třídy, přídavná jména představují hodnoty atributů a slovesa představují většinou aktivity).
- Identifikují se třídy zobecňování objektů se stejnými atributy (+ zkoumá se možnost uspořádat třídy hierarchicky podle dědičnosti) a vztahy.
- Stanoví se integritní omezení na hodnoty jednotlivých atributů a případně na typy atributů.
- Vyhotoví se seznam nabízených a požadovaných služeb pro všechny metody (přehled toku zpráv)
- Implementují se metody (teprve po jednoznačném vymezení všech funkcí systému)

Hlavní rozdíly mezi RDM a ODM:

| | RDM | ODM |
|---|---|---|
| 1 | ▪ relační tabulka<br>▪ jeden záznam<br>▪ manipulace s atributy záznamu | ▪ množina objektů<br>▪ jeden objekt<br>▪ přenos a zpracování zpráv |
| 2 | normalizace relací (dekompozice) vede k rozptýlení popisu vlastností složitého objektu do mnoha tabulek | spojuje jednotlivé složky pomocí odkazů |
| 3 | záznamy relací jsou omezeny na jednoduché datové typy | složité strukturované datové entity - objekty, které lépe vystihují prvky reálného světa |
| 4 | manipulace s hodnotami atributů záznamů | operace posílání zpráv poskytuje větší možnosti |
| 5 | každá tabulka musí mít identifikační klíč (ten nemusí odrážet požadavky zadání) | zabezpečuje identifikaci objektů vlastními systémovými prostředky (OID) |
| 6 | při zpracování dotazů dochází často k získávání údajů z několika tabulek ⇒ narůstá čas potřebný k vyhodnocení dotazu | ke spojování množin dochází v daleko menší míře; dotazovací konstrukce lze díky polymorfismu aplikovat i na množiny obsahující různé typy objektů |

Pozn.: RDM za určitých podmínek představuje zvláštní případ ODM.

Produkty: Objectivity, Versant, POET, CACHÉ, db4o, Ozone, GOODS, XL2, ZODB, Prevayler

Bariéry rozšíření objektových databázových systémů:

- neochota vývojářů a jejich klientů k přechodu od tradičního relačního přístupu k objektovému
- nedostatek kvalifikovaných vývojářů
- nízká podpora standardů
- nízká podpora dotazovacích jazyků
- neexistence mechanismu pro řízení přístupu k datům

Původní text (dle mého "mimo mísu")

Objektový (konceptuální model)

- představuje statický model reality (businessu)
- popisuje z čeho je realita složena a jaké jsou základní (podstatné/statické) složky (objekty) a vazby mezi nimi.
- Akce (metody) a algoritmy, vázané k objektům v jejich životních cyklech, zde mají význam též statický (jsou podřízeny statickému - pohledu).
- K popisu lze využít specifický diagram – Diagram tříd (Class Diagram - základní diagram jazyka UML).
- Model (entitních, business, analytických ) objektů (podstata struktury reality) sleduje základní stavební kameny, z nichž se realita (problémová doména) skládá.

Diagram tříd

- Původní strukturální přístup k analýze IS spočíval v rozdělení systému na funkční a datovou část (např. DFD - Data Flow diagramy a ER - Entity Relationship model).

- Přínosem byla funkční hierarchická dekompozice – psaní programu shora dolů a datové konceptuální modelování.
- Rostoucí složitost systémů (magická hranice 1000 entit a 10000 funkcí) znemožňuje soudržnost datové a funkční vrstvy.
- Objektový přístup čelí složitosti systému tím, že třída (objekt) jako nositel (funkční) odpovědnosti (dovednosti), plně odpovídá za svá data.
- Objekt má svou identitu, vlastnosti, chování a odpovědnost. Síla odpovědnosti spočívá v tom, že je nedělitelná – žádný jiný objekt nemůže odpovědnost sdílet – dělit se o ni, plést se do ní.
- Modelování tříd a objektů je klíčová aktivita objektově orientovaného vývoje.

Třída, Objekt

- **Třída** - popis množiny objektů sdílejících stejné vlastnosti (atributy), chování (operace/metody) a vztahy.
- **Objekt** - instance třídy (chybně se pojem třída a objekt volně zaměňují).

Definice – J. Rumbauh: objekt je diskrétní entita s jasně definovaným rozhraním, které zapouzdřuje stav a chování.

- Třídu si můžeme představit jako razítko, objekty jsou pak otisky tohoto razítka, které vidíme na papíře.
- Při návrhu třídy neuvažujeme o konkrétním naplnění atributů, pouze určíme jejich název a typ. Teprve při vzniku instance objektu se atributům přiřadí skutečné hodnoty.
- Třída je jednoznačně určena svým názvem (v příslušném názvovém prostoru – balíčku). Pro třídu je možno definovat vlastnosti - atributy (Attribute) a chování - operace (Operation).
- Hledání tříd, jejich atributů a kompetencí - vyberme z reality objekty, kandidáty pro zobecnění na třídy a prověřme jejich vhodnosti:
  - Potenciální třída je smysluplná, pokud je nezbytná pro funkci systému.
  - Potenciální třída je dostatečně stabilní a invariantní vůči vnějším změnám např. technologie, legislativy apod.

**Hledání tříd na základě analýzy podstatných jmen a sloves.**

Analyzujeme jazyk problémové domény, např. text sebraných požadavků. Podstatná jména a jejich spojení mohou označovat třídy nebo atributy. Slovesa mohou označovat odpovědnosti, chování tříd.

Pozor na skryté, utajené třídy, které nejsou v textu uvedeny.

Klasifikace, zařazení do třídy přenáší význam na formální objekt, je nositelem sémantiky. Klasifikace je jedním z nejdůležitějších způsobů, jímž lidé uspořádávají, vnímají, chápou okolní svět. Existuje mnoho způsobů jak klasifikovat okolní svět, proto je analýza tak náročná.

**Atributy**

**Definice atributů**

Atribut určuje vlastnosti objektu, je nositelem informace o objektu.

Atributy popisují hodnoty (stavy) udržované v jednotlivých objektech.

Objekty jsou vymezeny (popsány) množinou atributů.

Atributy popisují vlastnosti objektů, které potřebujeme k dosažení daného cíle. Reálný objekt ve své nekonečné složitosti nelze vymezit omezenou množinou atributů. Základní problém analýzy IS - výběr rozumného množství relevantních atributů.

S atributy mohou manipulovat výhradně služby daného objektu.

Klíčovou otázkou je zodpovědnost určitého objektu za uchování informací. Hledání atributů je řízeno otázkami: Jak je objekt popsán v kontextu zodpovědností daného systému. V jakých stavech se může objekt v průběhu svého životního cyklu nacházet

**Specifikace atributů**

Atribut je definován: jménem, typem (formátem) viditelností (veřejný – public, soukromý – private a

chráněný – protected).

Každý atribut pečlivě pojmenujte. Volte názvy, které jsou běžné v aplikační oblasti a jsou rozumné délky a pevné struktury. Ke každému atributu připojte vysvětlující text.

Hledejte omezující podmínky pro hodnoty atributů. Omezení se vztahují na: formáty, rozsah, výčet přípustných hodnot, přesnost implicitní hodnoty, požadavek na nastavení výchozích hodnoty atributu prevalidační a postvalidační podmínky – podmínky, které musí být splněny před a po změně hodnoty atributu, za jakých podmínek je povolen přístup k atributu (např. v závislosti na hodnotách ostatních atributů) závislost atributů, viz datové modelování, jak změna jednoho atributu ovlivňuje hodnoty jiných – závislých atributů, viz normalizace relačního modelu.

**Identita objektu**

Objekt je vedle svého stavu a chování jednoznačně určen, je jedinečný, má identitu, atribut, který jej jednoznačně identifikuje mezi všemi ostatními objekty dané třídy, má své ID. Atribut zajišťující identitu, se v datovém modelování nazývá primární klíč.

Čtenář (číslo čtenáře, jméno, adresa, kontakt) Kniha (isbn, autor, titul) Exemplář (číslo exemp, datum nákupu) Exempláře knihy se liší inventárním číslem a sledujeme u nich datum nákupu.

Problematika volby primárního klíče,

**Umístění atributů**

Ve třídách, které jsou vázány dědičností (generalizace) umístíme atribut do co nejvyšší třídy, ve které atribut platí pro všechny její generické podtřídy (specializace).

**Hledání tříd, doporučení**

Každá třída by měla mít 3-5 klíčových odpovědností První extrém - není dobré, když existuje velké množství malých tříd Druhý extrém – není dobré mít velké třídy Nezavádějte „funktiody“ – pro jednotlivé funkce systému nezavádějte třídy Vyhýbejte se stromům dědičnosti s mnoha úrovněmi

**<u>Vazby – relace mezi třídami</u>**

**Vazba asociace (Association)**

Vazba asociace mezi třídami je vyjádřením abstraktního vztahu mezi objekty (instancemi tříd). Asociace říká, že objekty mají mezi sebou přímý vztah, že o sobě ví.

Zaměstnanec pracuje v daném oddělení, mohu se ptát: V jakém oddělení pracuje zaměstnanec, mohu získat seznam všech zaměstnanců v oddělení. Vazba je nositelem významu – sémantiky, odpovídá – mapuje požadavky kladené na systém. Je trvalejšího charakteru.

Asociace je společný typ vazby pro:

- agregaci, vyjadřující vztah mezi celkem a částí
- prostou asociaci, vyjadřující prostou objektovou referenci.

Vazba asociace je specifikována řadou vlastností, z nichž některé jsou vázány přímo k vazbě asociace (například název), ostatní k zakončením vazby (například role). Podrobné určení vlastností až v okamžiku návrhu – specifikace návrhových tříd a jejich vazeb má „implementační“ důsledky.

Vazbu asociace lze zavézt jako orientovanou (Navigability), přičemž neorientovaná vazba je považována za obousměrnou (dva jednosměrné vztahy).

Třídy v asociaci mohou vůči sobě vystupovat v rolích (Role) (Objednávka – Zaměstnanec, Zaměstnanec vystupuje ve vztahu k objednávce v roli Prodejce). Každá strana asociace má své jméno – roli. Role popisuje vlastnost, funkci třídy „viděné“ z druhé strany.

V asociaci lze určit násobnost vazby, (kardinalitu) multiplicitu, která vyjadřuje počet možných vazeb objektů tříd v asociaci (0, 0..1, 0..*, 1, 1..*, *, M..N, …).

**Násobnost vazby** definuje, kolik může k jednomu objektu, tj. k jedné instanci třídy A na jedné straně vztahu existovat minimálně (parcialita) a maximálně (kardinalita) objektů ze třídy B na druhé straně vztahu a obráceně.

- Standardně je vazba asociace implementována zavedením atributu třídy - role pro zachycení objektové reference (množiny referencí pro parcialitu 0..*) na objekty druhé třídy. Implementace vazby – objekt si sebou nese reference na asociované objekty.
- S vazbami je třeba šetřit.
- Na rozdíl implementace v relačním datovém modelu se jedná o explicitní vyjádření vazby. V relačním modelu se pro vyjádření vazby používají tzv. cizí klíče – implicitní implementace vazby.

Další vlastnosti vazeb související s implementací vazby (mimo UML, CASE):

Každý konec asociace, vazby se nazývá role. Pro roli můžeme definovat řadu vlastností.

- Asociativní třída (Association Class) je vazba asociace, která je rozšířena přiřazením třídy pro zachycení informací nutných pro úplnou specifikaci této vazby.
- Asociativní třída se používá například v případě oboustranně násobné vazby N:M. Asociativní třída nemá vlastní identitu, identitu přejímá od „asociovaných“ tříd
- Vztah mezi třemi a více prvky popisují vícenásobné asociace (N-ary Association). Pro vyjádření vícenásobné asociace se používá element modelu asociativní třída.

**Vazba agregace (Aggregation)**

Agregace je vyjádřením abstrakce vztahu mezi objekty (instancemi tříd), který odpovídá vztahu celku a části .

Agregace je speciálním případem asociace. Jazyk UML rozlišuje mezi dvěma typy agregací:

- prostou agregací (Simple Aggregation)
- kompozicí (Composition).

Vazba kompozice je silnější než vazba prosté agregace, jedná o vlastnictví částí celkem. Pokud je celek existenčně (tj. svou logikou) závislý na částech a nemůže bez nich fungovat, jedná se o kompozici – pokud se bez nich obejde, jedná se o agregaci.

**Agregace**: Profesoři - Katedra - zruším katedru, profesoři zůstanou, mohou fungovat bez katedry


**Kompozice**: Fakulta - katedra, zruším fakultu, katedra je bezvýznamná


**Vazba generalizace (Generalization)**

Vazba generalizace je vyjádřením vztahu mezi obecným elementem (Parent) a specifickým elementem (Child), který je konzistentní s obecným elementem a přidává k jeho definici další informace, je tedy bližší specifikací (specializací) obecného elementu.

Vazba generalizace mezi třídami je vyjádřením vlastnosti dědičnosti, jedné ze základních vlastností objektově orientovaného přístupu.

Jazyk UML povoluje vyjádřit vícenásobnou dědičnost zavedením více vazeb generalizace.

**Vazba závislosti (Dependency)**

- umožňuje znázornit jistou závislost mezi elementy modelu.
- je určena svým názvem a obvykle se používá s určitým stereotypem, který blíže specifikuje formu závislosti, zavádí její typ.
- je znázorněna orientovanou přerušovanou čárou, kde orientace je vyjádřena šipkou ve směru závislosti.

Závislost obvykle vzniká pouze dočasně pro potřeby poskytnutí služby klientskému objektu a poté

tato vazba zaniká (implementační rozdíl od asociace).

Změna jednoho (nezávislého) elementu ovlivní druhý (závislý) element.

## Chování objektů, předávání zpráv

Objekt poskytuje služby prostřednictvím operací (metod).

Rozhraní objektu je množina operací, které nabízí objekt k použití pro jiné objekty (nebo externí agenty). Objekty jsou známy jiným objektům pouze prostřednictvím svého rozhraní. Objekt má i své vnitřní – interní operace, které slouží k udržení vnitřní konzistence (stavu) objektu.

Objekt může poskytovat více rozhraní – mít více rolí, podle kontextu ve kterém se nachází. Stejně jako v reálném světě člověk vystupuje v různých rolích podle toho, v jakém kontextu se právě nachází (v zaměstnání se nachází v roli pracovníka, doma manželem, v automobilu řidičem )

Objekty tak odbourávají nevýhodu strukturálních metod, spočívající ve vzájemné izolaci funkční a datové vrstvy.

Objekty spolupracují proto, aby společně mohly vykonávat funkce poskytované systémem, viz modely spolupráce. Operace určující chování jsou definovány a rozpoznávány svojí signaturou – názvem, seznamem parametrů a návratových hodnot. Objekt přijme zprávu a vykoná operaci, jejíž signatura je shodná se signaturou zprávy.

Pro lepší pochopení myšlenky OODB: http://www.linuxexpres.cz/business/objektove-databaze

## Objektové databáze

Stejně tak jako lidé postoupili od strukturálního programování k objektovému, tak si řekli, že by nebylo od věci neukládat data do tabulek a relací, ale do objektů tak, jak s nimi pracujeme přímo v programu. Bylo by přeci velice pěkné, když bych mohl objekt tak, jak ho mám, prostě uložit do databáze a o nic víc se nemuset starat.
Nemusel bych přemýšlet nad strukturami tabulek (tak aby dodržovaly „dobré mravy“ dané normami) a tvořit ruční INSERTy a SELECTy, jen bych databázovému stroji přes nějaké API řekl, načti mi uživatele s číslem 451, a dostal bych ho se všemi atributy naplněnými.
A přesně takto objektové databáze fungují. Místo tabulek jsou zde uloženy přímo objekty, včetně svých vlastností, a místo řádků se ukládají samotné instance objektů. Každý takto vložený objekt je jednoznačně identifikován svým OID, které na logické úrovni odpovídá ukazateli do virtuální paměti počítače a stejně tak se chová (při přesunu v paměti se změní i OID). Není tedy potřeba vytvářet primární klíče na objektech ani normalizovat databázi.
Objektové databáze také nabízejí využití možností vícenásobné dědičnosti, zapouzdření a polymorfizmu. Navíc vlastnosti (datové hodnoty) objektů nemusí být jen primitivního typu, ale mohou být dále strukturované jako například objekt (pomocí reference), množina nebo seznam. Pojem zapouzdření znamená, že každý objekt obsahuje nejen datové hodnoty (vlastnosti), ale i funkce, které definují, jak je možné s těmito vlastnostmi zacházet.
Polymorfizmus umožňuje objektům zastupovat své potomky (ve smyslu dědičnosti) při volání metod. Program nemusí znát přesný typ objektu, který volá, ale ten se zjistí až za běhu a zavolá se metoda na správné třídě.
Pro vytváření nových tříd v databázi je definován nový speciální jazyk ODL (Object Definition Language). Nicméně v dnešní době se využívá vlastností pokročilých programovacích jazyků, jako je reflexe. Například v db4o stačí zavolat metodu set na objektu databáze, předat jí jako parametr obyčejný objekt a zbytek už si zařídí knihovna sama.

```
void storePilot (string pilotName, int pilotsPoints)
{
  Pilot pilot1 = new Pilot(pilotName, pilotsPoints);
  db.Set(pilot1);
}
```

Pro načítání dat z objektové databáze existuje jazyk OQL (Object Query Language). Jak je vidět už podle názvu, jeho syntaxe je velice blízká SQL. V následujícím příkladu si povšimněte, že odpadla potřeba spojovat tabulky, protože vše je dostupné přes objektové vazby:

```
SELECT o.customer_id.get_name(), o.room_id.id
FROM orders o
```

```
WHERE o.check_day(o.room_id.id, datefrom, dateto) = 1;
```

Další velice zajímavou možností je vytvořit dotaz pomocí QBE (Query By Example). Princip tohoto přístupu spočívá v tom, že databázi předáme částečně naplněný objekt a ta nám ho dohledá ve svém zdroji a doplní ostatní vlastnosti. Přesněji řečeno vrátí nějaký kontejner naplněný objekty, které původnímu objektu odpovídají. Uvedu zde jeden ilustrační příklad:

```
Pilot retrievePilotByName (string pilotName)
{
 Pilot proto = new Pilot("Michael Schumacher", 0);
 IObjectSet result = db.Get(proto);
 if (result.HasNext())
   return (Pilot)result.Next();
 else
   return null;
}
```

From <http://www.linuxexpres.cz/business/objektove-databaze>

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>