VLIV RELAČNÍHO A OBJEKTOVÉHO DATOVÉHO MODELU NA FUNKČNOST A TVORBU DATABÁZOVÝCH SYSTÉMŮ

Vojtěch Merunka

Katedra informatiky, PEF VŠZ v Praze, 165 21 Praha-Suchdol
tel.: (02) 338 2272, fax.: (02) 338 2274, e-mail: merunka@pef.vsz.cz

Anotace:

V článku je diskutován vliv rozdílných vlastností objektového a relačního datového modelu na funkčnost a tvorbu databázových informačních systémů se zaměřením na porovnání dotazovacích možností z pohledu příslušných algebraických operací.

Summary:

The influence of different properties of the object-oriented and relational data model to the functionality and development of database systems is described with focus at comparison of query possibilities of algebraic operations in both data models.

Klíčová slova:

Objektově orientovaná analýza a návrh, objektový datový model a objektová algebra, relační datový model a relační algebra, databázové systémy.

Key words:

Object-oriented analysis and design, object-oriented data model and object algebra, relational data model and algebra, database systems.

¬¬¬

Objektově orientovaný přístup (OOP) je založen na teorii objektového datového a výpočetního modelu. Kořeny OOP lze vysledovat přibližně od konce 60. let především v konstrukci simulačních systémů (systém Simula-67) a dále v experimentech souvisejících s výzkumem nových architektur počítačů, operačních systémů a uživatelských rozhraní (Smalltalk) a s výzkumem v oblasti reprezentace znalostí.

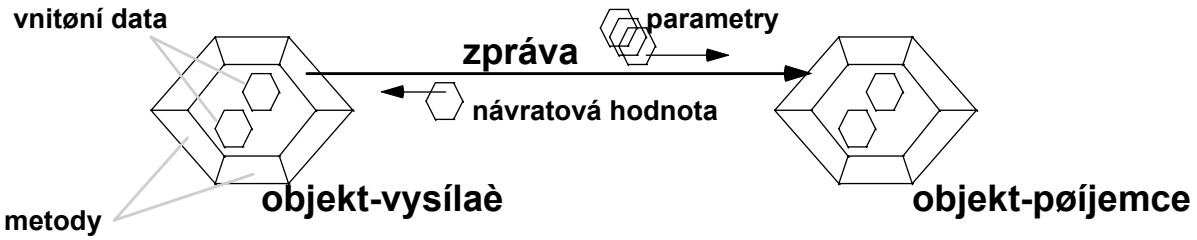
V OOP se na rozdíl od klasického pojetí operuje s objekty, které popisují jak datovou, tak i procesní stránku řešené problematiky. Objekt je určitá jednotka, která modeluje nějakou část reálného světa a z funkčního pohledu víc odpovídá malému kompaktnímu programu, než jedné proměnné příslušného datového typu. Objektový systém je potom souborem takovýchto vzájemně interagujících malých programových celků.

Datová povaha objektu je dána tím, že objekty se skládají z příslušných **vnitřních dat - složek**, což jsou v opět nějaké jiné objekty. Funkční povaha každého objektu je dána tím, že každý objekt má jakoby okolo svých vnitřních dat obal či zeï, která je tvořena množinou

malých samostatných částí kódů, jenž jsou nazývány **metodami**. Metody slouží k tomu, aby popisovaly, co daný objekt **dokáže dělat** např. se svými složkami (objekty). Se složkami nějakého objektu lze manipulovat (číst, nastavovat, měnit) pouze pomocí kódu nějaké metody tohoto objektu. Každý objekt dovoluje provádět jen ty operace, které **povoluje** jeho množina metod.

V objektovém modelu výpočtu pracujeme pouze se dvěma možnými operaci s objekty. První z nich je **pojmenování** nějakého objektu jiným objektem. Druhou je tzv. **poslání zprávy**. Zpráva představuje **žádost o provedení** operace - metody nějakého objektu. Součástí zprávy mohou být tzv. **parametry zprávy**, což jsou vlastně data - objekty, které představují **dopředný datový tok** (ve směru šíření zprávy) směrem k objektu přijímajícímu danou zprávu.

Poslání zprávy má za následek provedení kódu jedné z metod objektu, který zprávu přijal, tak tento zmíněný kód také vesměs dává nějaký **výsledek** v podobě nějakých dat - objektů, které představují **zpětný datový tok** ve směru od objektu - příjemce zprávy k objektu - vysílači zprávy (tj. v opačném směru k šíření zprávy). Vzhledem k možnostem kódů metod se výsledky po poslaných zprávách **neomezují** pouze na hodnoty jednotlivých složek objektů, protože jsou dány libovolně složitým výrazem příslušné metody nad množinou všech složek objektu sjednocenou s množinou parametrů zprávy.



Další vlastností je takzvané **dědění mezi objekty**, které nám umožňuje definovat vnitřní strukturu a metody nových objektů pomocí definic jiných již existujících objektů. To, že se při popisu a programování nových objektů odkazujeme na již existující objekty, nám umožňuje tyto nové objekty definovat pouze tím, jak se od stávajících objektů **liší**, což přináší kromě obrovských úspor při psaní programů také možnost vytvářet "nadtypy" a "podtypy" mezi objekty, vyčleňovat společné vlastnosti různých objektů atp. Umožňuje-li systém přímo (ne zprostředkovaně přes jiné objekty) dědit od více než jednoho objektu, potom podporuje tzv. **vícenásobné dědění**.

Velmi důležitým pojmem v oblasti OOP je také pojem **třídy objektů**. Máme-li v systému velké množství objektů, které mají sice různá data, ale shodnou vnitřní strukturu a shodné metody, potom je výhodné pro tyto objekty zavést jeden speciální objekt, který je nazýván **třída**, a který pro všechny tyto objekty dohromady shromažïuje popis jejich vnitřní struktury a jejich metody. Takto popsané objekty jedné třídy jsou nazývány **instancemi** této

třídy. V některých systémech jsou třídy implementovány pouze jako abstraktní datové typy a ne jako objekty, což kromě jiných omezení znamená, že nemohou být vytvářeny či modifikovány za chodu programu.

V objektových systémech se můžeme setkat ještě s jedním důležitým druhem hierarchie mezi objekty - tzv. **závislostí** mezi objekty. V závislosti objektů rozeznáváme dva typy objektů: řídící objekty - tzv. **klienty** a řízené objekty - tzv. **servery**. Žádá-li nějaký klient provedení nějaké operace od serverů, tak posílá zprávu, neboť zpráva je i zde jedinou možností, jak spustit nějakou operaci. Na rozdíl od standardního poslání zprávy, kdy je třeba znát příjemce zprávy, v případě závislých objektů klient nepotřebuje znát svoje servery, protože oni sami jsou povinni svého klienta sledovat a zprávy od něj zachycovat. Zprávu, která je signálem pro servery, tedy objekt klient posílá bez udání příjemce. Tato hierarchie bývá úspěšně využívána při tvorbě grafických uživatelských rozhraní a při tvorbě komplikovanějších simulačních modelů.

Z naznačených vztahů lze odvodit následující vlastnosti objektového modelu:

a) Pro objektový model a jeho funkčnost je základní a postačující hierarchie skládání objektů do sebe, která úzce souvisí s ideou zapouzdření lokální informace, členěním systému do modulové struktury a sjednocením datové a funkční stránky.

b) Hierarchie dědění, třídně-instanční a závislosti objektů jsou na sobě formálně nezávislé a je možno je považovat za odvozené vlastnosti objektového modelu. Za určitých podmínek (dostatek paměti, odhlédnutí od efektivnosti, srozumitelnosti implementace atp.) je možné udržet funkčnost jakékoli objektové aplikace při vynechání některé (nebo i všech) z těchto odvozených hierarchií. Toto tvrzení mj. dokazuje i absence vztahu závislosti v systémech C++, ObjectPascal aj.

c) Na základě vnějšího pozorování funkčnosti objektového modelu bez nahlédnutí do vnitřní struktury systému není možné identifikovat přítomnost žádné odvozené objektové hierarchie uvnitř modelu.

d) Polymorfismus objektů je vlastnost, která není závislá pouze na existenci dědičnosti mezi objekty. Objekty mohou být polymorfní i bez vztahu dědičnosti.

Objektový a relační datový model se od sebe výrazně liší ve své funkčnosti i v architektuře. Předložme nyní hlavní formální rozdíly mezi oběma modely:

1) Relační model je oproti mnoha možnostem objektového modelu založen pouze na množinách n-tic, přičemž pokud jednotlivé prvky těchto n-tic mohou být jen atomickými hodnotami, jedná se o tzv. **normalizovaný relační model**. Popsané vlastnosti sice nebrání v relačním modelu za prvky n-tic dosazovat strukturované hodnoty (další n-tice nebo celé tabulky), ale tato možnost se v praxi používá jen velmi ojediněle v tzv. NFNF resp. NF^2 (non-first normal form) databázích.

2) Objektový model kromě datové (statické) informace podporuje oproti relačnímu modelu navíc i práci s procedurální (dynamickou) informací.
3) Operace poslání zprávy u objektového modelu má větší funkčnost než jí odpovídající atributová funkce u relačního modelu.
4) Vzájemně si odpovídající operace obou modelů mají odlišné vlastnosti na změny protokolů objektů (resp. prvků) i s-protokolů množin objektů (resp. prvků).
5) Operace kartézského součinu a s ní související operace spojení objektů resp. prvků odpovídá obecné algebraické definici pouze u objektového modelu. Spojením prvků (n-tic) relačního modelu v kartézském součinu totiž nevzniká uspořádaná dvojice spojovaných prvků, protože spojení dvou n-tic není dvojice těchto n-tic, ale opět n-tice složená ze všech dílčích hodnot ze spojovaných n-tic). Operaci spojení objektů lze jako objekt - dvojici objektů realizovat v souladu s obecnou algebrou.
6) Objektové množinové operace nejsou omezeny na množiny obsahující prvky (objekty) stejné struktury jako jim odpovídající objektové operace. Jsou-li objektové množiny složeny z různých objektů (dáno jejich potokoly), potom dochází při množinových operacích ke změnám jejich s-protokolů, přičemž tyto změny (např. u operace selekce) mohou být **využity v návrhu** modelované aplikace.
7) Pojetí identity u objektů se nekryje s pojetím identity u záznamů. Identita objektu je určena kromě vnitřních hodnot i vnějšími vazbami daného objektu. Identita záznamu je určena pouze jeho vnitřními hodnotami.

Přistoupíme-li na transformační zobrazení vztahu relačního a objektového datového modelu podle uvedené tabulky,

relační datový model	**objektový datový model**
relační tabulka	datová množina
entita, jedna relace z tabulky (záznam)	objekt
atribut, atributová funkce	zpráva

tak můžeme vyjádřit jak relační datový model pomocí objektového formálního aparátu, kde největší sémantický rozdíl v transformační tabulce překonává mapování atributu relační entity na zprávu posílanou objektu. Pojem atributu záznamu je tu z hlediska objektového modelu chápán jako operace získání určité hodnoty pomocí atributové funkce a ne jako jeho statická hodnota sama o sobě, jak je jinak běžné u relačního datového modelu.

objektová selekce

$Select(\mathbf{A_i}, \pi) = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A_i} \ \pi(\mathbf{a})\}$
$Select(\mathbf{A_i}, \pi) \subseteq \mathbf{A_i}$.
$SProt(\mathbf{A_i}) \subseteq SProt(Select(\mathbf{A_i}, \pi))$

objektová kolekce

$Collect(\mathbf{A_i}, \lambda) = \{ \lambda(\mathbf{a}) ; \mathbf{a} \in \mathbf{A_i}\}$
$| Collect(\mathbf{A_i}, \lambda) | = | \mathbf{A_i} |$
$SProt(Collect(\mathbf{A_i}, \lambda))$
$= Prot(\lambda(\mathbf{a}_1)) \cap ... \cap Prot(\lambda(\mathbf{a}_n))$

objektová projekce

$Project(\mathbf{A_i}, \{\mu_k\})$
$= \{\mathbf{a}.\mu_1 \oplus \mathbf{a}.\mu_2 \oplus ... \oplus \mathbf{a}.\mu_m ; \mathbf{a} \in \mathbf{A_i} \}$
$| Project(\mathbf{A_i}, \{\mu_k\}) | = | \mathbf{A_i} |$
$SProt(Project(\mathbf{A_i}, \{\mu_k\})) \not\subset SProt(\mathbf{A_i})$

objektové spojení dvou množin

$Join(\mathbf{A}_1, \mathbf{A}_2, \mu_1, \mu_2, \otimes)$
$= \{\mathbf{a}_1 \oplus \mathbf{a}_2 ; \mathbf{a}_1 \in \mathbf{A}_1 \wedge \mathbf{a}_2 \in \mathbf{A}_2 \wedge \mathbf{a}_1.\mu_1 \otimes \mathbf{a}_2.\mu_2\}$
$SProt(Join(\mathbf{A}_1,\mathbf{A}_2,\mu_1,\mu_2,\otimes))$
$\supseteq SProt(\mathbf{A}_1) \cup SProt(\mathbf{A}_1)$

objektové sjednocení dvou množin

$\mathbf{A}_1 \cup \mathbf{A}_2 = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A}_1 \vee \mathbf{a} \in \mathbf{A}_2\}$
$SProt(\mathbf{A}_1 \cup \mathbf{A}_2) = SProt(\mathbf{A}_1) \cap SProt(\mathbf{A}_2)$

objektový průnik dvou množin

$\mathbf{A}_1 \cap \mathbf{A}_2 = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A}_1 \wedge \mathbf{a} \in \mathbf{A}_2\}$
$SProt(\mathbf{A}_{1,2}) \not\supset SProt(\mathbf{A}_1 \cap \mathbf{A}_2)$

relační selekce

$Select(\mathbf{A_i}, \pi) = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A_i} \wedge \pi(\mathbf{a})\}$
$Select(\mathbf{A_i}, \pi) \not\supset \mathbf{A_i}$.
$SProt(\mathbf{A_i}) = SProt(Select(\mathbf{A_i}, \pi))$

relační kolekce

nemá ekvivalent v relační algebře

relační projekce

$Project(\mathbf{A_i}, \{\alpha_k\})$
$= \{\mathbf{a}.\alpha_1 \oplus \mathbf{a}.\alpha_2 \oplus ... \oplus \mathbf{a}.\alpha_m ; \mathbf{a} \in \mathbf{A_i} \}$
$| Project(\mathbf{A_i}, \{\alpha_k\}) | = | \mathbf{A_i} |$
$SProt(Project(\mathbf{A_i}, \{\alpha_k\})) = \{\alpha_k\}$

relační spojení dvou množin

$Join(\mathbf{A}_1, \mathbf{A}_2, \alpha_1, \alpha_2, \otimes)$
$= \{\mathbf{a}_1 \oplus \mathbf{a}_2 ; \mathbf{a}_1 \in \mathbf{A}_1 \wedge \mathbf{a}_2 \in \mathbf{A}_2 \wedge \mathbf{a}_1.\alpha_1 \otimes \mathbf{a}_2.\alpha_2\}$
$SProt(Join(\mathbf{A}_1,\mathbf{A}_2,\alpha_1,\alpha_2,\otimes))$
$= SProt(\mathbf{A}_1) \cup SProt(\mathbf{A}_1)$

relační sjednocení dvou množin

$\mathbf{A}_1 \cup \mathbf{A}_2 = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A}_1 \vee \mathbf{a} \in \mathbf{A}_2\}$
$SProt(\mathbf{A}_1 \cup \mathbf{A}_2) = SProt(\mathbf{A}_1) = SProt(\mathbf{A}_2)$

relační průnik dvou množin

$\mathbf{A}_1 \cap \mathbf{A}_2 = \{ \mathbf{a} ; \mathbf{a} \in \mathbf{A}_1 \wedge \mathbf{a} \in \mathbf{A}_2\}$
$SProt(\mathbf{A}_{1,2}) = SProt(\mathbf{A}_1 \cap \mathbf{A}_2)$

Tyto uvedené rozdíly mají praktický dopad nejen na rozdílnou funkčnost objektových a relačních systémů, ale především i na **proces návrhu** těchto systémů, protože

1) Objektový model podporuje teoeticky neomezeně složité a strukturované datové entity - objekty v různých vzájemných hierarchiích, které mohou snáze korespondují s "reálnými objekty" v zadání, resp. konceptuálním modelu aplikace. Kromě jednoduchých typů objektů (texty, čísla, grafika, zvuky), je možné vytvářet i nové typy objektů jako kombinace existujících. Každý objekt se navenek projevuje jako jedna celistvá entita. Tento fakt se příznivě projevuje na dotazovacích a vůbec funkčních možnostech aplikace.

Relační databáze je omezena jen na n-tice z jednoduchých přípustných datových typů, jakými jsou např. čísla či texty (u moderních systémů i grafika a zvuky), přičemž každý složitější údaj či struktura musí být v relačním modelu **transformován** na soustavu n-tic sestavených z přípustných typů.

2) U relačního datového modelu jsou v entitách databáze uloženy pouze statické vztahy - data. Dynamická stránka je záležitostí programu pracujícího s takovouto databází. Objekty, se kterými pracuje objektový model, kromě statických údajů - dat obsahují v databázi i kódy metod - tj. dynamickou stránku uložených dat nezávisle na manipulujícím programu.
3) Údaje v relačních tabulkách (tj. složky jednotlivých n-tic) jsou přímo přístupné přes příslušné domény, jejichž názvy tvoří protokol daných n-tic. Údaje v objektovém modelu jsou ukryté v objektech a namísto techniky přímého přístupu do vnitřní struktury se pracuje s takovými protokoly objektů, které vymezují každému objektu množinu přípustných zpráv. Mezi protokolem objektu a jeho vnitřní datovou strukturou nemusí být přímá souvislost. Zprávami posílanými objektům lze získávat i takové údaje, které nejsou přímou součástí struktury daných objektů, neboť mohou být výsledkem libovolného výpočtu (viz. ad 2).
4) Vzhledem k 3) je u relačního modelu nutné, aby všechny údaje v rámci jedné domény zachovávaly nějaký stejný datový typ a všechny n-tice jedné tabulky stejnou strukturu, což prakticky znamená, že všechny navrhované množiny v relačním systému musejí být navrhovány již od konceptuální fáze analýzy pouze z prvků stejného typu. Množiny z prvků různého typu podobně jako vztahy **m:n** je nutno rozkládat. Vztahy **1:n** je nutné u normalizovaných relačních modelů implementovat pomocí propojení dvou tabulek, kde vazba (propojení pomocí klíčů) vždy směřuje od entit "**n**" směrem k entitám "**1**".
5) U relačního modelu během činnosti DBMS, tj. během zpracovávání některých dotazů, dochází k dočasnému spojování jednotlivých relačních tabulek, neboť každá složitější struktura byla při své tranformaci do relačního datového modelu rozdělena do více n-tic uložených v různých relačních tabulkách. Při dotazu proto musejí být komplexní údaje jdoucí mimo rámec údajů uložených v jedné tabulce spojovány (pomocí klíčů) z údajů více tabulek, což představuje časově i uživatelsky náročnou operaci.

   U objektového modelu k spojování množin dochází jen v záměrných netriviálních případech, neboť možnosti skládání objektů nejsou teoreticky nijak omezené.
6) U relačního modelu je pro potřeby zajištění identity navrhovat kromě běžných atributů tzv. klíče záznamů, které se nemusí vždy krýt s potřebami zadání. Na klíčích relačních záznamů, které jsou de facto pouze implementačním prostředkem (abstrahované ukazatele do paměti), závisí celý aparát výrazových prostředků dotazovacího jazyka.

U objektového modelu (z důvodu komplexnějšího pojetí objektové identity) stačí v návrhu modelovat pouze ty složky, které vždy odrážejí nějaký informační aspekt ze zadání. **Klíče jako implementační prostředek jsou výhradně vnitřní záležitostí systému**.

7) U relačního modelu může být bez ohledu na konceptuální schéma a zadání problému vztah **1:n** implementován pouze ve směru od záznamů typu **n**, které obsahují klíče na záznamy typu **1**, což vyplývá z normalizačních pravidel a vztahy typu **m:n** musejí být implementovány tzv. vazebními tabulkami, což má přímý vliv i na sémantiku dotazů v relačním dotazovacím jazyce, kde je třeba se v některých případech podřizovat skutečnému směru příslušné vazby.

Objektový model umožňuje implementaci vazeb mezi objekty v souladu s konceptuálním modelem ve směru od celku k části bez ohledu na případnou kardinalitu.

P o z n á m k a :

Uvedené vlastnosti objektového datového modelu jsou v praktických systémech plně implementovány pouze u tzv. čistě objektově orientovaných databázových systémů, mezi které patří např. systémy Gemstone, Artbase, OODBMS, O2, Versant aj. Úlohu dotazovacího jazyka v těchto databázích plní nejčastěji klony jazyka Smalltalk, který je zároveň i aplikačním programovacím jazykem. (*Klasické relační databáze používají jako dotazovací jazyk verze jazyka SQL a jako aplikační jazyk verze jazyků 4GL, přičemž je třeba překonávat z důvodu odlišných výrazových prostředků a nutnosti vzájemné konverze datových typů u zmíněných jazyků tzv. impedanční problém*) Jako "objektově orientované" jsou však v dnešní době záměrně označovány z komerčních důvodů i nejrůznější verze databázových systémů (Paradox, MS Access apod.) založených na relačním datovém modelu, přičemž se vesměs jedná o dovybavení těchto systémů různými uživatelskými rozhraními či o podporu nestandardních datových typů (grafika, zvuky, sdílená data s jinými programy, ...) v relačních tabulkách.

Z naznačených rozdílů lze odvodit, že **relační datový model může za určitých podmínek představovat zvláštní případ objektového datového modelu**, přičemž rozbor vzájemných souvislostí mezi modely se jeví jako velmi prospěšný pro návrh hybridních systémů, které se v současné době v praxi vedle čistých objektově orientovaných systémů stále více používají. Nejčastěji se jedná o případ kombinace klient-server relačního databázového prostředí (server) s front-end aplikacemi (client) vytvářenými pomocí objektově orientovaných programovacích jazyků (OOPL).

Literatura

V. Merunka; přednášky z předmětu Jazyky umělé inteligence na PEF VŠZ v Praze, 1991-1994.

V. Merunka; přednášky z předmětu Objektově orientované programování na FEL ČVUT v Praze, 1992-1994.

V. Merunka; Objektově orientovaná analýza a design; ve sborníku 20. ročníku semináře Programování'93, Ostrava 1993.

J. Polák, R. Knott, V. Merunka; Principles of the O-O Paradigm, published as the tutorial book on EastEuroOOP'93 conference, November Bratislava.

V. Merunka, J. Polák; OOA & OOD - Teaching and Application Issues, in proceedings of 20th International ASU Conference, Průhonice 1994.