# Objektová databáze - od modelu k implementaci

**Tomáš Hruška**[1]

Fakulta elektrotechniky a informatiky, Vysokého učení technického v Brně, Ústav informatiky a výpočetní techniky, Božetěchova 2, 612 66 Brno, Česká Republika

**Abstract**

This paper describes experience of the design and implementation of the object-oriented database model. The first part discusses some reasons for the design of new object-oriented based information systems. The second part defines the term *object-oriented database* in the context of a large number of various object-oriented systems. The third part describes the object-oriented database model. Two well known standards are introduced and some new extensions of them are shown. The last shows main features of the language, compiler and internal interpreted code.

## 1. ÚVOD

Mnoho organizací uvažuje o zavedení nového informačního databázového systému. Jejich starší informační systémy, stavěné často horkou jehlou a ve značném chvatu podnikatelské euforie začátku devadesátých let, překračují postupně svůj zenit a stávají se nezvládnutelnými a nemodifikovatelnými molochy. Jsou totiž často realizovány autory amatéry v amatérských podmínkách, buďto zcela bez podpory databázového systému (soubory operačního systému ovládané v Pascalu se zdály býti dostatečnými), s podporou různých elementárních databázových systémů, nebo v nejlepším případě relační technologií.

Zdá se, že nyní se situace mění. Pokusy o vytváření levných informačních systémů vlastními silami se postupně vytrácejí. Finančně nenáročný přístup již nebývá dále určujícím kritériem, neboť zkušenosti ukazují, že kvalitní informační systém je pro podnik nezbytný a šetřit se nevyplácí. V mnoha případech už rovněž není peněz nedostatek jako v pionýrských začátcích.

Chuť řešit problém zavedení informačního systému moderní technologií, kvalifikovanými odborníky a pokud možno s rozumnou perspektivou vývoje je logickým důsledkem těchto trendů. Častým navrhovaným řešením je využití *objektově orientovaného databázového systému*. Použití této technologie je zdůvodněno často výbornými zkušenostmi s vývojem jiných nedatabázových objektově orientovaných systémů, případně je rozhodnutí ovlivněno pozitivními referencemi na podobný způsob vývoje. Dalším důvodem je nutnost plánovat řešení s perspektivou mnoha let. Proto je nutné vybrat technologii, která není již v současnosti zastaralá a tuto mnohaletou perspektivu má.

Problémy objektové technologie trápí ovšem více ta pracoviště, která se zabývají vytvářením programových systémů, resp. odborníky firem, kteří jsou za informační

[1] Práce na návrhu a realizaci prototypu objektově orientovaného modelu je podporována grantem GAČR 102/96/0986 *Objektově orientovaný databázový model*

systém odpovědní. Uživatelé informačních systémů očekávají zvýšení kvality práce a pojem objektové orientovanosti u nich stojí na stejné úrovni zájmu jako pojem relačního databázového prostředí.

Tento článek je proto psán spíše z hlediska vývojového pracovníka, nežli uživatele. Je stručným souhrnem úvah, kterými bylo třeba se zabývat při implementaci objektově orientovaného databázového systému.

## 2. VYMEZENÍ POJMU OBJEKTOVĚ ORIENTOVANÉHO DATABÁZOVÉHO SYSTÉMU

Z důvodů uvedených v úvodu je přívlastek *objektově orientovaný* hlavně velice dobrá obchodní nálepka. Proto se těžko hledá databázový (a nejen databázový) systém, který by objektově orientovaný nebyl. Dá se říci, že každý touto nálepkou označený systém jistou dávku objektové orientovanosti nese. U databázových systémů se můžeme setkat zhruba s následujícími nabízenými variantami:

- *objektově orientované prostředí* pouze pro vytváření aplikačních programů (databázový stroj není objektový),
- objektově orientované *rozšíření relačního databázového systému*,
- objektově orientovaný *databázový* systém.

### 2.1 Objektově orientované prostředí pro vytváření aplikací

Půjde obvykle o systém pro realizaci aplikačních programů (aplikací) objektově orientovanou technologií. Implementačním jazykem bude pravděpodobně C++. Nejčastěji půjde o vytváření aplikací v grafickém prostředí oken. Na databázovém stroji a jeho vlastnostech u této skupiny z hlediska naší klasifikace příliš nezáleží. Obvykle jde o relační databázi ovládanou příkazy SQL. Výsledkem je z uživatelského hlediska moderní uživatelské rozhraní. Tvůrce řeší nadále klasické problémy převodu databázových typů jazyka SQL (seznamy) na typy implementačního jazyka, tzv. problém *impedance*. Výhodou je snadné vytvoření moderního vzhledu systému. Často lze tohoto prostředí použít i pro vytváření nedatabázových aplikací. Perspektiva systému je dána perspektivou databázového prostředí, nad nímž prostředí pracuje. Je-li styk s databází prováděn příkazy standardního relačního rozhraní ODBS, lze vytvořit aplikace spolupracující prakticky se všemi momentálně komerčně dodávanými databázemi.

### 2.2 Objektově orientované rozšíření relačního databázového systému

Půjde obvykle o standardní relační databázový systém s tím, že základní typy jsou rozšířeny o typ, do něhož je možné uložit rozsáhlé binární údaje předem nedeklarované délky, tzv. *bloby - binary large objects*. Databázový systém pak musí být tedy schopen ukládat tyto záznamy *proměnné délky*. Předpokládá se, že do blobů bude možné ukládat multimediální a jiné rozsáhlejší údaje. Termín *object* v názvu blobů bývá rovněž představován jako příslib objektové technologie. Chybějí zde všechny základní atributy objektů, tj. zapouzdření, dědičnost ap. (viz kap. 3). Proto použitý termín objekty je pouze homonymem a o objektovou technologii ve skutečnosti nejde.

Tuto metodu je možné kombinovat s předchozí (čl. 2.1) a tím realizovat graficky komunikující aplikace nad databázemi s multimediálními položkami. Případně je možné do blobů ukládat i binární strukturu reprezentující skutečný komplexní objekt. Databázový systém však není schopen takovou strukturu obhospodařovat, neboť ji není schopen rozpoznat co do její vnitřní struktury a veškeré operace s takovými objekty pak zůstávají na aplikačních programech. Nelze pak např. využívat obecný prohlížeč databáze a neexistují žádné vlastnosti databázového stroje podporující objektovou technologii.

## 3. OBJEKTOVĚ ORIENTOVANÝ DATABÁZOVÝ SYSTÉM

Třetí uvedenou variantou je *plnohodnotný objektově orientovaný databázový systém*. Takový systém musí poskytovat prostředky pro definici databázového modelu s objektově orientovanými rysy. Tyto rysy musí být plně podporovány databázovým systémem. Jde zejména o následující rysy:

- existence objektů s identifikací,
- komplexní objekty,
- zapouzdření,
- třídy a typy,
- dědičnost,
- přetížení operátorů, upřesňování, pozdní vazba a polymorfismus.

Význam pojmů z předchozího seznamu nebudeme podrobně diskutovat. Byly popsány např. v [6]. Zvolíme-li takový databázový systém pro realizaci informačního systému, změní se technologie realizace zhruba v následujících rysech.

Model, jímž popisujeme modelované universum, bude organizován jako *typový strom*, resp. orientovaný acyklický graf daný dědičností (jednoduchou, resp. vícenásobnou) tříd objektů. Objekt je definován jako kartézský součin hodnot jeho atributů.

Objekt má jednoznačnou *identifikaci* unikátní v celém prostoru, v němž se může vyskytovat. Je zaveden a užíván datový typ množina (seznam, kolekce). Atributy objektu jsou libovolného (i strukturovaného) typu. Je-li atributem jiný strukturovaný typ, lze dosáhnout libovolné komplexnosti objektu.

Neomezenost datového typu atributu dovoluje ukládání *multimediálních údajů*. Atributem může být (na rozdíl od relačního modelu) i *vztah* k jednomu, či více objektům. Vztahy tudíž nevznikají dynamicky v době dotazu, ale jsou součástí datového prostoru. Vzhledem k tomu, že atributem objektu nejsou pouze elementy pevné délky, ale i délek proměnných (např. množiny), je nutné při implementaci řešit situace, kdy objekt v průběhu existence mění svoji délku.

Kromě datových atributů a vztahů mohou objekty obsahovat i atributy popisující procesy (algoritmy). Tím se objekt stává autonomní jednotkou, v níž jsou zapouzdřena jak data, tak procesy.

Proces zobrazení reálné skutečnosti do objektového modelu je přímočarý, není nutné rozbíjet významové jednotky do nepřirozených struktur z důvodů sémantické

vyjadřovací schopnosti např. relačního modelu. Údaje spolu související jsou uloženy v rámci jediného objektu, což slibuje efektivní implementaci. Vztahy jsou součástí modelu, což eliminuje používání drahé relační operace spojení.

Pokud tyto argumenty budeme považovat za podstatné a rozhodneme se pro objektově orientovaný databázový systém, ocitneme se před problémem, který vhodný objektově orientovaný databázový systém zvolit. Odpověď je poměrně snadná. Neexistuje žádný natolik ověřený plnohodnotný systém uvedených vlastností, který by bylo možno použít s tím, že jeho výrobce jednak implementoval úplný objektově orientovaný model a zároveň zaručuje existenci a údržbu systému po celou předpokládanou existenci navrhovaného informačního systému, tj. alespoň deset až dvacet let.

Nemáme-li kapacity pro vývoj vlastního systému, nezbývá, nežli zvolit některý starší obvykle pouze částečně objektově orientovaný systém nebo počkat. V současnosti jsou již k dispozici některé systémy první vývojové vlny objektových databází (např. $O_2$). Ty však v sobě nesou stopy přechodu z relační technologie. Jsou k dispozici i beta verze systémů (Poet, Jasmine), které deklarují použití úplného objektového modelu. Jejich plné nasazení je pravděpodobně otázkou blízké budoucnosti.

Pokud se rozhodneme pro vývoj vlastního databázového prostředí, musíme si být vědomi rozsáhlosti projektu. Je nutné především

- zvolit, resp. navrhnout vlastní objektový model,
- navrhnout jazyk, který definuje jak data, tak procesy,
- vytvořit překladač tohoto jazyka,
- navrhnout cílový jazyk překladu, což je v případě definice dat katalog struktury databáze a v případě manipulací nejlépe interpretovatelný mezijazyk přenositelný na více platforem,
- implementovat přenositelný interpret tohoto mezijazyka (jazyk obsahuje i dotazy, je třeba počítat s jejich optimalizací),
- navrhnout vlastní databázový stroj a to nejlépe tak, aby bylo možno použít jako jeho část stávající relační databáze, jejíž obsah bude chtít potenciální uživatel dále používat,
- navrhnout způsob komunikace objektové databáze s uživatelem a toto uživatelské rozhraní implementovat jako grafické tak, aby bylo možné jej použít pro různé klientské stanice,
- implementovat generátory tiskových sestav a formulářů,
- resp. navrhnout a implementovat mnoho dalších komponent systému.

Všechny tyto komponenty by měly být organizovány tak, aby jejich počet a jejich způsob spolupráce v systému byl flexibilní. V neposlední řadě je třeba počítat s tím, že v době dokončení bude nutné přesouvat objekty v rozsáhlých sítích. V současnosti je nepominutelná možnost spolupráce s protokolem *http* a možnost komunikace s databází přes Internet.

## 4. MODEL

Již volba definitivního tvaru modelu, který hodláme implementovat, není jednoduchá. V případě implementace relační databáze je jednoznačnou normou relační model dat. Taková norma v oblasti objektově orientovaných databází neexistuje. Jsou činěny pokusy takovou normu vytvořit a to dvojím způsobem:

- rozšířit stávající *SQL* o objektově orientované rysy,
- realizovat objektovou databázi jako nadstavbu nad obecným prostředkem práce s objekty.

První z přístupů je zřejmě vhodnou cestou, jak stávající relační systémy rozšířit na objektové. Je však otázkou, zda tento postup neponese jako přítěž svůj původní účel. Toto úsilí zřejmě směřuje k vytvoření tzv. *objektově relačních databází* [5]. Jejich představitelem je v současnosti pravděpodobně systém Versant, jehož tvůrcem je profesor Stonebreaker.

Druhý přístup je jiný. Jeho reprezentantem je v současnosti pracovní skupina ODMG 93. Jako výchozí stav zvolila objektové prostředí CORBA vyvíjené skupinou OMG. Toto prostředí je rozšířeno pro práci s objekty v databázích. Návrh této normy je pojmenován stejně jako pracovní skupina ODMG 93 a v současnosti je vydána knižně verze 1.2.

### 4.1 Rozšíření SQL

Objektově orientované SQL odráží snahu přenést výhodné vlastnosti relačního jazyka SQL do objektově orientovaného prostředí. V části dotazovací využívá syntaktickou stavbu jazyka SQL (**SELECT** ... **FROM** ... **WHERE**). K tomuto konceptu však doplňuje možnost pojmenovávat objekty klauzule FROM a obohacuje klauzuli **SELECT** následujícím způsobem:

- objekty jsou zpřístupňovány užitím klíčových hodnot,
- objekty mohou být v dotazu označovány proměnnými a těchto proměnných je možné v rámci dotazu dále používat,
- V klauzulích **SELECT** a **WHERE** mohou být použity operace.

### 4.2 Nadstavba nad modelem OMG

Cílem iniciativy ODMG-93 je poskytnout uživateli normu umožňující uživateli objektově orientovaného databázového systému psát přenositelné aplikace, tj. aplikace, které mohou běžet na více nežli jediném objektovém databázovém systému. Popis modelu, vazba na programovací jazyk, manipulační a dotazovací jazyk jsou navrhovány jako portabilní. Norma je navrhována tak, aby výsledné produkty dovolovaly interoperabilitu těchto produktů s *využitím OMG Object Request Brokeru CORBA*.

Základní rysy modelu ODMG-93 jsou následující:

- stav objektu je určen hodnotami *vlastností*, těmito vlastnostmi mohou být buď *atributy* nebo *vztahy* mezi objektem a jedním nebo více jinými objekty,

- chování objektů je definováno množinou operací, které mohou být prováděny nad objektem daného typu. Operace kromě standardních parametrů umožňují definici výjimek, které charakterizují stav operace po ukončení některým z nestandardních stavů. Tato metoda specifikace chybových stavů je převzata z OMG.

V definici třídy objektů je se mohou objevit následující rysy:

- *nadtřídy* a *podtřídy*. Model povoluje vícenásobnou dědičnost. Kromě toho je v normě naznačena možnost upřesňování datových typů vzhledem k nadtřídě, což může být vhodné např. při zužování intervalu.
- *klíče* slouží k unikátní identifikaci objektů hodnotou některých atributů. Je možné využívat jak *jednoduchých*, tak *složených* klíčů.
- metody jsou specifikovány *signaturou operace*. Signatura definuje jméno operace, jméno a typ parametrů, jméno a typ návratové hodnoty, případně výjimku, kterou může operace generovat.
- atributy jsou popsány *signaturou atributu*. Zde je zadáno jméno a datový typ atributu.
- vztahy jsou specifikovány *signaturou vztahu*. Vztahy mohou být dvojího typu, 1:1 a 1:N. Z důvodu kontroly konzistence je možné ke každému vztahu definovat jeho inverzní obraz. Inverzní vztahy jsou automaticky udržovány systémem. Nad vztahem je možné popsat i lineární uspořádání.
- model obsahuje několik generátorů množinových typů nazvaných *collection*. Jsou to *set*, *bag*, *list* a *array*. Těmito generátory je možné organizovat vztahy typu 1:N. *Set* je klasická množina a *bag* multimnožina. *List* je uspořádaný seznam s možností přímého přístupu a *array* dynamické pole.
- každá třída může tvořit tzv. *extent*. Extentem rozumíme vztahy typu 1:N, které nejsou atributem jiného objektu. Extenty obsahují všechny objekty dané třídy a všech jejích následníků. Extent je automaticky udržován databázovým systémem. Extenty jsou přístupné z prostředí mimo model a jsou v objektově orientované databázi jedním ze základních přístupových metod do databáze.

### 4.3 Další rysy objektového databázového modelu

Publikované objektově orientované modely krátce popsané v předchozích odstavcích nejsou z hlediska objektového databázového modelu ideální. V obou případech není manipulační jazyk definován jako výpočetně úplný a tudíž není možné se vyhnout při jeho použití tzv. *impedanci*. Impedancí rozumíme rozdílný typový prostor manipulačního jazyka databázového modelu a jiného obvykle univerzálního jazyka, v němž je implementován zbytek databázové aplikace.

Jde zejména o problém s ovládáním typu množina, kolekce, resp. seznam, které jsou základním modelovacím databázovým prostředkem, přičemž nejsou standardním typem univerzálních jazyků, jako např. C++. To si vynucuje trvalé převody databázových typů na nedatabázové v průběhu činnosti databázové aplikace a vytváření pomocných struktur typu *kurzor*, aktuální prvek ap. pro práci se seznamy. V této souvislosti je třeba zvážit, zda je výhodnější použít obecně známý univerzální jazyk, do něhož je databázový jazyk vnořen a připustit impedanci nebo navrhnout jazyk jako celek s tím, že bude typově konzistentní, avšak nikoliv obecně známý.

Další problém vyplývající z existence množiny jako základního modelovacího prostředku v objektových databázích souvisí s tím, že základní nedatabázový objektový model tento pojem nezná, tudíž metody nad objekty jsou zde vytvářeny zásadně nad jediným výskytem objektu a rovněž neexistují prostředky pro práci s přirozenými podmnožinami objektů. Tento problém bude podrobněji diskutován v čl. 4.3.1.
Obdobně není v základním objektovém modelu diskutována možnost existence objektu, který není pouze jediné třídy, ale v daném okamžiku může mít tříd více. Ukazuje se, že z hlediska databázových aplikací je použití takových vícetypových objektů velmi praktické. Tento problém bude diskutován v čl. 4.3.2.

*4.3.1 Množiny, metody, agregáty*

V základním objektovém modelu není možné deklarovat atributy množin objektů. Pokud jsou deklarovány metody nad objektem, jejich implicitním atributem (nazývaným často *this*) je objekt samotný. V objektově orientovaném databázovém modelu je množina základní datovou strukturou. Datový typ množina je hodnotou vztahu 1:N, může být výsledkem dotazu ap. Ukazuje se, že by bylo výhodné mít možnost používat pro datový typ množina metody, jejímž implicitním atributem *this* není objekt, nýbrž množina objektů. Takové speciální metody provádějí operace nad množinou jako celkem, vyhledávají v množině, provádějí sumační, resp. agregační výpočty.

Metody nad izolovanými objekty a nad množinami jsou navzájem převoditelné. Pokud by byla metoda deklarovaná pro jediný objekt použitá pro argument typu množina, pak je provedena pro všechny prvky množiny. Naopak, pokud je metoda deklarovaná pro množinu použitá pro jediný objekt, pak je provedena pro jednoprvkovou množinu vytvořenou tímto objektem.

Speciálními metodami deklarovanými nad množinami jsou *agregáty*. Standardními agregáty jsou obvykle číselné hodnoty vypočtené nad jedinou položkou všech prvků množiny, např. *součet*, *průměr*, *maximum*, *minimum*, *standardní odchylka* ap. Další agregáty lze deklarovat jako obecným postupem deklarace metod nad množinami.

Jak již bylo řečeno, takto definované metody lze použít nad množinami, které jsou v modelu nejčastěji k dispozici jako členové vztahu 1:N nebo jako výsledek dotazu. Takto získané množiny je dále možné členit na přirozené podmnožiny (faktorizovat na základě definované ekvivalence) a tyto podmnožiny trvale udržovat. To lze provádět dvojím způsobem:

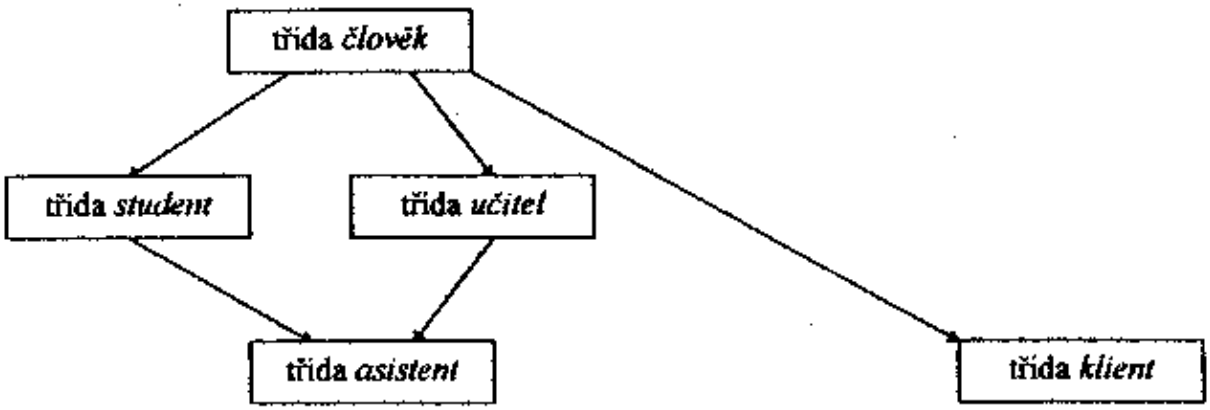
- *automaticky* na základě hodnoty objektu nebo
- *manuálně* na základě uvážení uživatele.

Automatické udržovaní přirozených podmnožin je možná definicí posloupnosti výrazů nad objektem. Objekty, pro něž výrazy mají tutéž hodnotu, jsou zařazeny do téže třídy ekvivalence (faktorové množiny). Posloupnost takto definovaných výrazů vytváří hierarchický rozklad na množině.

Podobně lze hierarchický rozklad vytvářet ručně a tuto strukturu průběžně udržovat.

*4.3.2 Vícetypové objekty*

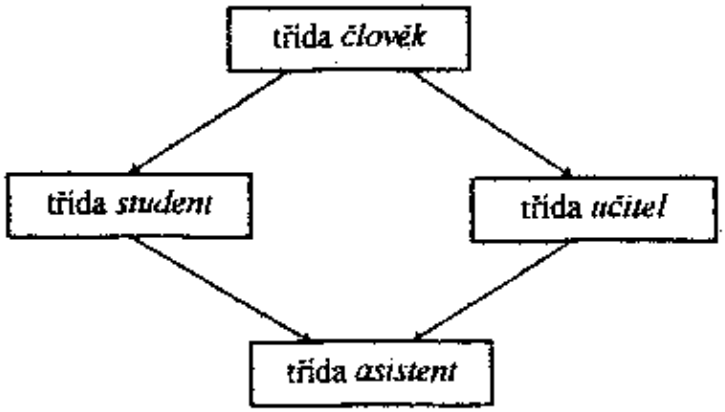
Klasické objekty užívané např. v C++ a v ostatních objektově orientovaných programovacích jazycích, ať již s dědičnosti jednoduchou nebo vícenásobnou, mohou být vždy pouze jediné třídy. Přesněji řečeno je při vytvoření určena jediná třída, která je jednou pro vždy datovým typem objektu. Vytvářený objekt je této třídy a všech jejich předchůdců.




Obr. 1. Typový orientovaný acyklický graf

Předpokládejme např. typový acyklický graf dědičnosti zobrazený na obr. 1.

Předkem všech tříd v typovém grafu je třída *člověk* reprezentující základní informace o libovolném člověku. Třídy *student* a *učitel* jsou následníci třídy *člověk*. Třída *asistent* reprezentuje studenta, který je zaměstnán již v průběhu studia. Třída *klient* modeluje člověka, který je klientem knihovny. V klasickém objektovém modelu může být konkrétní objekt pouze těch tříd, které jsou na cestě od třídy v kořeni typového stromu k dané třídě, s níž byl objekt vytvořen. Např. objekt třídy *asistent* je současně objektem tříd *student*, *učitel* a *člověk*. Podobně objekt třídy klient je současně objektem třídy *člověk*. Pro konkrétní objekt má jeho typový acyklický orientovaný graf tvar diamantu s jediným vrcholem a jediným listem. Např. pro předchozí případ je diamant zobrazen na obr.2.




Obr. 2. Diamant pro konkrétní objekt

Pokud ve třídě *člověk* jsou uložena všechna základní data o člověku, např. jméno, příjmení, bydliště ap., může být obvyklé, že tento člověk se stane klientem knihovny. Pokud bude použit klasický objektový model, nesmí být objekt třídy *klient* současně objektem třídy *asistent*. Tuto situaci lze řešit v klasickém objektovém modelu pouze

zavedením umělé třídy, která bude následníkem všech sjednocovaných tříd. Toto řešení je však nesystémové a zavádí třídy, které nemodelují žádnou skutečnou entitu. Použití vícetypových objektů je proto v současnosti předmětem výzkumu [14].

## 5. KATALOG, JAZYK A PŘEKLADAČ

V realizované databázi je uloženo několik úrovní informací:

- *data* uložená v databázi,
- *metadata* - data reprezentující informaci o tom, jaký tvar mají data uložená v databázi (často nazývaná *katalog*) a
- *meta²data* - data reprezentující informaci o tom, jaký tvar mají metadata.

Meta²data jsou obvykle pevná a neměnná. Vyjadřují tvar informací uložených v katalogu. Metadata v katalogu jsou popisem tvaru modelu jak po stránce datové tak procesní a jsou výsledkem návrhu databáze. Metadata ukládá do databáze tvůrce její formy. Data jsou vlastním obsahem databáze a do databáze je ukládá uživatel.

Je otázkou návrhu databázového systému, zda jsou data a metadata uložena v téže databázi, či zda jsou metadata uložena ve specializovaném tvaru se specializovaným způsobem přístupu. Je potřeba zvážit, zda bude metadata možné měnit za chodu databáze a bude-li uživateli umožněn přístup k metadatům.

Model je v databázi uložen v metadatech, jeho obsah v datech. Metadata je třeba naplnit k tomu, aby bylo možné pracovat s daty. Pokud je k dispozici vývojový systém (např. vhodný case systém), jehož výstupem jsou přímo metadata, nebylo by teoreticky nutné používat specializovaný jazyk pro popis tvaru údajů, i když jej obvykle k dispozici máme. Pro popis manipulačních algoritmů a dotazů se však návrhu jazyka nevyhneme. Překladem jazyka je plněn katalog dávkovým způsobem a tím je databáze připravena k činnosti.

## 6. VNITŘNÍ INTERPRETOVATELNÝ KÓD

Výkonné instrukce ovládacího jazyka budou ve většině případů prováděny na více počítačích. Uvažujeme-li koncepci *klient-server*, budou prováděny jak na *klientovi*, což mohou být počítače různých typů, tak na *serverech*, což jsou rovněž obecně různé platformy. Na všech těchto platformách mohou být různé reprezentace základních číselných typů. Rovněž je třeba předpokládat činnost systému v různých národních prostředích, což předpokládá různé kódování národních abeced. Použití dvoubytové reprezentace znaků (Unicode) zajistí možnost reprezentace libovolného znaku všech světových abeced, avšak vyžaduje dvojnásobný prostor pro řetězce.

Nutnost předpokládat činnost systému na více platformách a současně zvyšující se výkonnost počítačů směřují k překladu do obecného interpretovatelného kódu. Toto řešení umožňuje rovněž ukládat do databázových objektů i algoritmy, neboť takovýto mezikód je vlastně datovým typem funkce a procedury.

Nabízejí se různé typy vnitřních kódů. Nejpoužívanější je pravděpodobně *zásobníkový kód*. Standardní řešení zásobníkového kódu předpokládá uložení

operandů na vrcholu zásobníku, který je tvořen relativně neomezenou pamětí pro uložení operandů jak metod, tak všech operací. Použijeme-li jako operandy objekty neomezené obecnosti, dochází k implementačnímu problému způsobenému tím, že operand mění během výpočtu svoji velikost. To vyžaduje netradiční řešení zásobníku, z nichž jedno (pomocí pomocného zásobníku) je uvedeno např. ve [12].

**LITERATURA**


[1] ELLIS,M.,A.-STROUSTRUP,B.: The Annotated C++ Reference Manual, Addison-Wesley Publishing Company 1990, 453 stran

[2] BERTINO, E.-MARTINO,L.: Object-Oriented Database Systems (Concept and Architectures), Addison-Wesley Publishing Company 1993, ISBN 0-201-62439-7, 264 stran.

[3] LOOMIS, M.E.S.: Object Databases The Essential, Addison-Wesley Publishing Company 1995, ISBN –201-56341-X, 230 stran.

[4] CATELL, R.,G.,G. (ed.): The Object Database Standard: ODMG-93 (Release 1.2), San Francisco, Morgan Kaufmann 1996, 184 stran, ISBN 1-55860-396-4

[5] STONEBRAKER,M.,MOORE,D.: Object-relational DBMSs - The Next Great Wave, San Francisco, Morgan Kaufmann 1996, 216 stran, ISBN 1-55860-397-2

[6] HRUŠKA,T.: Objektově orientované databázové technologie, in: Sborník konference *Tvorba software 95*, Tanger Ostrava 1995, str. 113-124

[7] HUDGES,J.,G.: Object-Oriented Databases, Prentice Hall 1991, 280 stran

[8] BENEŠ, M., HRUŠKA, T., MÁČEL, M.: Nested Objects and Items of Variable Length. In Proceedings of 22nd Conference of the ASU, University Blaise Pascal of Clermont Ferrand, pp. 39-46, Clermont Ferrand 1996

[9] BENEŠ, M., HRUŠKA, T., MÁČEL, M.: Bootstrapping of an Object-oriented Database System In: Proceedings of 10th International Conference "Systems for Automation of Engineering and Research" and DECUS NUG Seminar, Varna 1996, pp.216-220

[10] HRUŠKA,T., KOLENČÍK, P.: Semantics of Object Identification in Object Oriented Database Model. In: Proceedings of Scientific Conference Electronic Computers and Informatics Faculty of Electrical engineering and Informatics of Technical University Košice 1996, pp.243-249

[11] HRUŠKA, T.: Object Models of Information Systems. In: Proceedings of MOSIS 96 Conference, Krnov, MARQ Ostrava and Department of Computer Science of Technical University Ostrava 1996, pp. 252-257

[12] ZENDULKA, J.: The Conception of The Method interpreter of The Object-oriented Database Management System. In: Proceedings of Scientific Conference Electronic Computers and Informatics Faculty of Electrical Engineering and Informatics of Technical University Košice 1996

[13] HRUŠKA, T.: Dědičnost. In: Sborník konference Tvorba software 96, MARQ Ostrava 1996, str.107-117

[14] BERTINO,E.,GUERRINI,G.: Object with Multiple Most Specific Classes, In: ECOOP 95 - Object-Oriented Programming, 9th European Conference Aarhus, Denmark, Proceedings in Lecture Notes in Computer Science 952, Springer Verlag 1995, str. 102-126