Objektově orientované databázové technologie

Tomáš Hruška

Objektově orientovaná technologie výstavby programových produktů je technologií modelování systémů. Modelovaný systém je chápán jako množina objektů se vzájemnou interakcí. Lidé jsou zvykli pohlížet na své okolí jako na objekty charakterizované jejich vlastnostmi, chováním a vzájemnými vztahy. Často také klasifikují objekty z hlediska jejich uspořádání, případně ekvivalence. Proto je pro ně snadné myslet tímtéž způsobem při návrhu modelu. Objektově orientovaný model je proto velmi často snadno pochopitelný a má přímý vztah k realitě. Objektově orientovaná technologie návrhu modelu získala v minulých letech značnou popularitu a byla v letech 1993 a 1994 předmětem mnoha referátů i na semináři Programování. Po přečtení sborníků z těchto let by opakování základních pojmů objektově orientované technologie bylo nošením dříví do lesa. Případný zájemce nechť laskavě je hledá např. v literatuře [15,16,17] ap.

1. Objektově orientované databázové systémy (ODBMS)

Na rozdíl od klasického použití objektově orientovaného modelu např. v jazyce C++ vyžaduje databázové nasazení další vlastnosti. Databázové objekty jsou uloženy na sekundární paměti a vyžaduje se efektivní a rychlý přístup k těmto objektům. Druhou vlastností, která s předchozí úzce souvisí, je schopnost objektu přetrvávat mezi jednotlivými běhy ovládacího programu (**persistence**). Podrobněji byly objektově orientované databáze diskutovány např. v [17].

Aby se uživatel databáze nemusel zabývat problémy souvisejícími s uložením informací, je mu k dispozici **systém řízení báze dat** (database management system - **DBMS**). Tento systém poskytuje uživateli logický pohled na databázi a zakrývá obvykle problém fyzického uložení informací. DBMS obvykle poskytuje následující rysy:

- **paralelismus** - dovoluje přístup více uživatelů simultánně k jediné společné databázi,
- **zotavení** - pokud se vyskytne softwarová či hardwarová chyba, DBMS umožní vrátit stav databáze do konzistentního stavu,
- **dotazovací prostředky** - DBMS obvykle podporuje snadný přístup k vhodně vybraným údajům z báze.

Relační databázové systémy (RDBMS) byly nasazeny na začátku sedmdesátých let. O deset let později již zcela ovládly databázový svět. V současnosti slouží jako téměř univerzální prostředek pohledu na údaje uložené v databázích. Dominujícími rysy RDBMS jsou jednoduchost a nezávislost údajů. Na druhé straně není tento model schopen postihnout sémantiku komplexních objektů. Takový objekt bývá často modelován v několika tabulkách, což činí přístup k údajům neefektivním (je totiž nutné spojit množství tabulek pro získání potřebné informace).

ODBMS si klade za cíl ukládat údaje objektu takovým způsobem, aby přístup k údajům tohoto objektu byl efektivní. Jedním z hlavních úkolů je minimalizace použití pomalých relačních operací spojení, které jsou v relačním modelu základní operací přístupu k údajům. Doposud užívané komerční ODBMS neužívají vlastní jazyk pro manipulaci s údaji, nýbrž užívají pro komunikaci standardních objektových jazyků (C++, resp. Smalltalk). Tím je vyřešen problém

impedance. Impedancí rozumíme nekonzistenci mezi datovými typy modelu a ovládacího jazyka. V takové situaci vzniká potom nutnost transformace typů, chceme-li např. příkazy jazyka zpracovat výsledek dotazu. Vzhledem k tomu, že vývoj ODBMS je dosud aktuální proces, pokusme se definovat vhodné vlastnosti ODBMS. Poznamenejme, že konečná úmluva mezi uživateli ODBMS v tomto směru je ještě značně vzdálena. Po ODBMS jsou většinou požadovány následující vlastnosti:

- **Komplexnost objektů**. ODBMS by měl podporovat modelování komplexních objektů.
- **Identita objektů**. Každý objekt má identitu nezávislou na jeho vnitřních hodnotách.
- **Zapouzdření**. ODBMS musí podporovat zapouzdření informací a operací v objektu.
- **Třídy**. Musí být podporován mechanismus strukturování a buďto ve formě typů nebo ve formě tříd.
- **Hierarchii**. Musí být podporována dědičnost.
- **Upřesňování**. ODBMS bude podporovat koncept upřesňování, resp. pozdní vazby (např. ve formě virtuálních metod).
- **Úplnost**. Ovládací jazyk bude schopen vyjádřit každou libovolnou operaci nad modelem. Impedance bude vyloučena.
- **Rozšiřitelnost**. Bude poskytnuta možnost přidávání nových typů a to i v čase běhu databázového systému, nikoliv pouze v čase návrhu modelu.

2. Některé objektově orientované databázové systémy.

Na rozdíl od komerčně uznávaných produktů (Ingres, Oracle, Gupta apod.) v oblasti relačních modelů, v oblasti objektově orientovaných modelů není situace zdaleka ustálená. Existují jak komerční produkty (ty však nejsou dosud široce používané), tak publikované výsledky výzkumných úkolů a grantů, které jsou často dostupné formou freeware. Zmiňme se o některých z nich. Následující přehled si nedělá ambice na úplnost. Jde spíše o ukázky produktů, které mohou být pro čtenáře zajímavé.

Nově navrhované objektově orientované databázové systémy vycházejí převážně ze dvou inspiračních zdrojů:

- **obohacování relační databázové technologie** o objektově orientované rysy,
- **obohacovaní objektově orientovaných programovacích jazyků** o databázové rysy.

2.1. Postgres

Postgres je produktem grantu řešeného na University of California profesorem Michaelem Stonebrakerem v minulých šesti letech. Jak už sám název napovídá, vychází z relačního databázového produktu Ingres. Model Postgresu lze považovat za rozšíření relačního modelu o dědičnost, komplexní objekty, verzování, zavedení času a práce s historickými objekty, uživatelské datové typy a ukládání operací do databáze. Postgres nepodporuje ukládání vztahů do databáze a předpokládá, že vztahy budou budovány v rámci dotazu. Klíčovou součástí je dotazovací jazyk Postquel. Postgres je dostupný na anonymním ftp `toe.cs.berkeley.edu`.

2.2. DAPLEX

DAPLEX [4] vychází z funkcionálního datového modelu. Základním pojmem funkcionálního modelu je entita. Entita má atributy a mezi entitami existují vztahy. Funkcionální datový model je charakteristický tím, že jak hodnoty položek modelu, tak vztahy mezi entitami jsou reprezentovány funkcemi.

Ukažme nyní příklad jednoduchého funkcionálního modelu informačního systému studentů.

```
DECLARE   Student( )                  ->>  ENTITY
DECLARE   Ident_číslo(Student)        ->   INTEGER
DECLARE   Jméno(Student)              ->   STRING
DECLARE   Adresa(Student)             ->   STRING
DECLARE   Pohlaví(Student)            ->   STRING
DECLARE   Ročník(Student)             ->   INTEGER
DECLARE   Kurz(Student)               ->>  Kurz

DECLARE   Kurz( )                     ->>  ENTITY
DECLARE   Číslo_kurzu(Kurz)           ->   INTEGER
DECLARE   Název(Kurz)                 ->   STRING
DECLARE   Ústav(Kurz)                 ->   Ústav
DECLARE   Učitel(Kurz)                ->   Učitel

DECLARE   Ústav( )                    ->>  ENTITY
DECLARE   Jméno(Ústav)                ->   STRING
DECLARE   Vedoucí(Ústav)              ->   Učitel

DECLARE   Učitel( )                   ->>  ENTITY
DECLARE   Jméno(Učitel)               ->   STRING
DECLARE   Kancelář(Učitel)            ->   INTEGER

DEFINE    Učitel(Student)             ->>
                                      Učitel(Kurz(Student))
```

Příkazy DECLARE deklarují nové funkce v modelu. Pokud jde o funkce bez paremetrů, jsou takto deklarovány nové typy entit. Příkaz DEFINE popisuje odvozenou funkci, která nepopisuje ani definici atributu ani vztahu. Vzhledem k jednoduché definici dat je jednoduchý i dotazovací a manipulační jazyk. Jeho sílu se pokusíme ukázat na následujícím příkladu:

- *Vytiskni jména všech studentů, které vyučuje učitel Jan Novák*

```
FOR EACH Student SUCH THAT
     FOR SOME Kurz(Student)
          Jméno(Učitel(Kurz(Student))) = 'Jan Novák'
PRINT Student(Jméno)
```

Cyklus FOR EACH provádí tělo cyklu pro všechny entity typu Student, které splňují podmínku zadanou v klauzuli SUCH THAT. Klauzule FOR SOME reprezentuje predikát, jehož splněním je entita typu Student zařazena do množiny, pro níž je cyklus prováděn.

Funkcionální datový model umožňuje elegantní zavedení dědičnosti. Předpokládejme např., že entity (nyní již z hlediska objektově orientovaného přístupu třídy) Učitel a Student z předchozího příkladu jsou podtřídami třídy Osoba, která je charakterizována společnou položkou Jméno. Tuto hierarchii lze ve funkcionálním modelu vyjádřit přirozeným způsob tak,

že oborem hodnot generátoru entity nebude obecná množina entit ENTITY, nýbrž ta třída, z níž je prováděno dědění.

```
DECLARE Osoba( )          ->>   ENTITY
DECLARE Jméno(Osoba)      ->    STRING

DECLARE Student( )        ->>   Osoba
DECLARE Učitel( )         ->>   Osoba
```

Model DAPLEX byl implementován jako objektově orientovaný databázový systém ADAM.

2.3. O2

O2 poskytuje procedurální objektově orientovaný jazyk pro manipulaci s údaji. Třída objektů je v tomto jazyce nazývána n-tice (**tuple type**). Pro vyjádření vztahu mezi n-ticemi se využívá funkcionálního zápisu. Např. podle:

```
add class       Zamestnanec
     type tuple (    cislo:          integer,
                     jmeno:          string,
                     oddeleni:       Oddeleni,
                     poschodi:       integer)


add class       Projekt
     type tuple (    cislo:          integer,
                     jmeno:          string,
                     rozpocet:       integer,
                     team:           set(Zamestnanec))
```

Objektem třídy `Zamestnanec` pak může být např. Einstein: (cislo=12345, jmeno=„Albert Einstein", oddeleni=*Matematika*, poschodi=3). Objektem třídy `Projekt` je např. Relativita (cislo=125, jmeno=„Relativita", rozpocet: 12000, team: (*Einstein, Minkowski, Lorentz*)). Identifikátory psané kurzívou přestavují odkazy na jiné objekty, resp. jejich množiny.

Metody jsou přidávány k jednotlivým třídám zadáním jejich *signatury*:

```
add method presun (noveposchodi: integer) in class Zamestnanec.
```

Jejich implementace může být provedena v jazyce C se speciálním rozhraním nazvaným CO2.

3. Normalizace

Snahy o sjednocení objektově orientovaného databázového modelu byly zatím málo úspěšné. Neexistence normy objektově orientovaných databázových systémů je jedním z hlavních omezení jejich širokého nasazení. Proto dochází k různým aktivitám směřujícím k normalizaci ODBMS. Kromě snahy o vytvoření objektově orientovaného SQL zasluhuje pozornosti aktivita několika amerických komerčních organizací o vytvoření normy ODBMS nazvaná ODMG [5].

3.1. Objektově orientované SQL

Objektově orientované SQL (OSQL) odráží snahu přenést výhodné vlastnosti relačního jazyka SQL do objektově orientovaného prostředí. OSQL 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 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.

Jako příklad uveďme dotaz v jazyce OSQL, jehož výsledkem jsou jména, oddělení a pracovní zařazení všech zaměstnanců pracujících na projeku číslo 121.

```
SELECT      Jmeno(e), Jmeno(Oddeleni(e)), Zarazeni(e,p)
FROM        Zamestnanec e, Projekt p
WHERE       Cislo(p) = 121
AND         p IN PracujeNa(e)
```

V rámci klauzule `FROM` jsou přiděleny proměnné e resp. p relacím `Zamestnanec` resp. `Projekt`. Těchto proměnných je pak využito při budování položek výsledku, které zde na rozdíl od klasického SQL jsou zapsány ve tvaru funkcí. Funkce `Jmeno` zpřistupňuje hodnotu jména z relace `Zamestnanec` apod. Funkce se vyskytují i v klauzuli `WHERE` (např. `PracujeNa`).

3.2. ODMG 93

Pracovní skupina ODMG byla založena Rickem Cattellem v létě roku 1991 a její stanovy jsou koncipovány tak, aby členové byli motivováni k intenzívní práci. Každý člen se např. zaváže, že minimálně jeden týden v měsíci bude věnovat výhradní práci na této normě. Nejde o akademickou, nýbrž o komerční aktivitu s cílem efektivního praktického využití. Ve zprávě ODMG - 93 z roku 1994 je uvedeno 19 členů této aktivity z mnoha renomovaných firem.

Návrh normy se skládá z následujících částí:

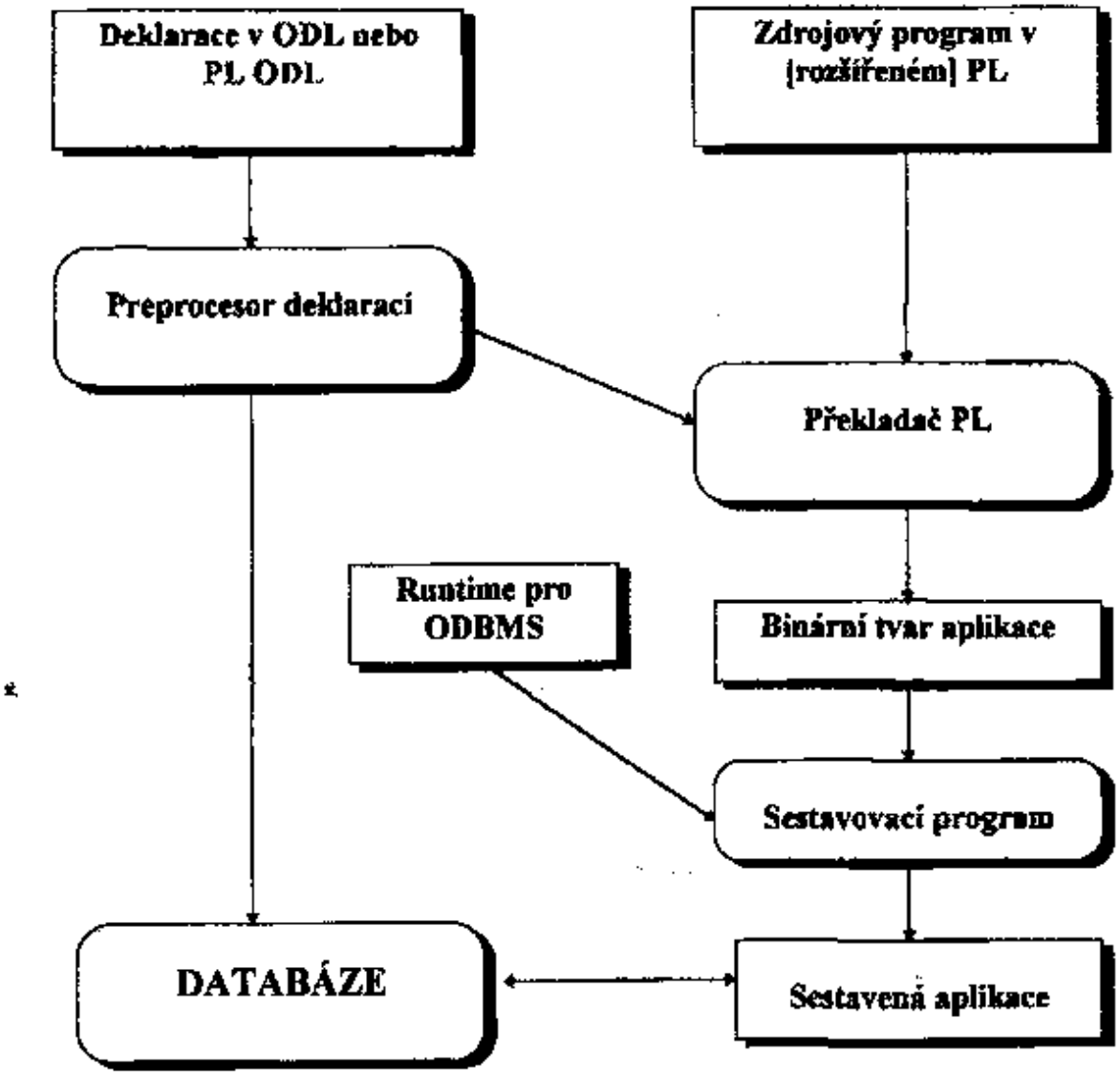
- objektový model.
- jazyk pro definici objektů (ODL),
- dotazovací jazyk OQL,
- vazba na jazyk C++,
- vazba na jazyk Smalltalk.

Typický produkt normovaného ODBMS je zobrazen na následujícím obrázku. Programátor vytvoří deklarace v jazyce ODL a zdrojový program aplikace. Zdrojový program je zapsán v programovacím jazyce (PL), kterým je C++, resp. Smalltalk. Tento jazyk je rozšířen o možnost ovládání databáze včetně transakcí a kladení dotazů. Deklarace databázového schématu je zapsána buďto ve speciálním definičním jazyce ODL nebo v rozšíření programovacího jazyka PL, nazvaném PL ODL. Deklarace a zdrojový program jsou přeloženy a sestaveny s knihovními podprogramy.

Základní rysy modelu ODMG jsou následující:

- Základní modelovanou entitou je **objekt**.
- Objekt má typ. Všechny objekty daného typu mají společné chování a společnou množinu hodnot, kterých mohou nabývat.

- Chování objektů je definováno množnou operací, které mohou být prováděny nad objektem daného typu.
- Stav objektu je určen hodnotami **vlastností**. Těmito vlastnostmi mohou být buď **atributy** nebo **vztahy** mezi objektem a jedním nebo více jinými objekty.



Uvažujme následující situaci:

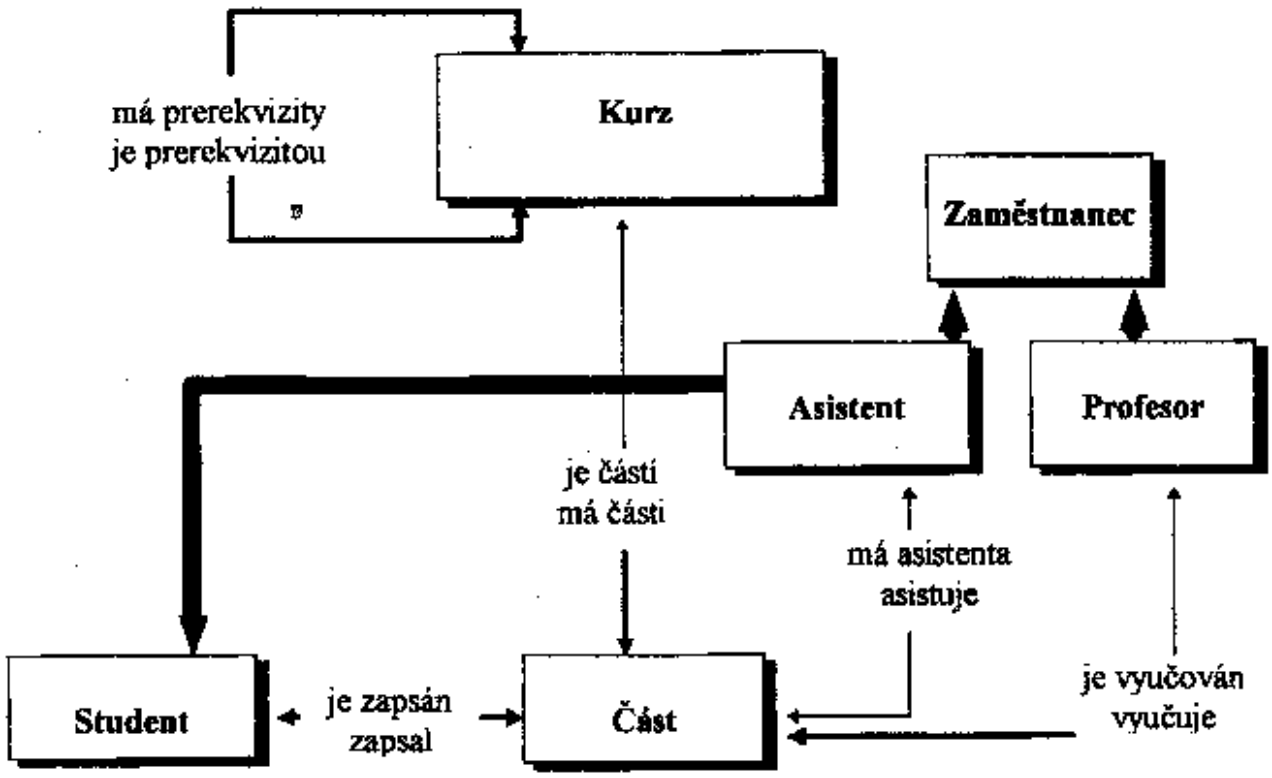
Entita `Kurz` představuje kurz na vysoké škole. Kurz má své jméno a evidenční číslo. Kurz může mít povinného předchůdce (prerekvizitu), což je opět kurz. Každý kurz může mít libovolný počet prerekvizit a může být prerekvizitou libovolného množství kurzů. Kurz může mít libovolný počet částí a `Část` může být částí jediného kurzu. Dalšími třídami modelu jsou `Zamčstnanec`, `Asistent` a `Profesor`. Poslední třídou je třída `Student`. Třída `Profesor` je podtřídou třídy `Zaměstnanec`. Třída `Asistent` (myslí se Teaching assistent z řad studentů) je podtřídou třídy `Zaměstnanec` i `Student` (zde je použita vícenásobná dědičnost).

Definice modelu v jazyce ODL může být následující (tenké čáry v následujícím obrázku představují vztahy 1:1, střední šířka vztahy 1:n a silné čáry bez popisu relací dědičnosti):

```
interface  Kurz
// záhlaví typu
(     keys        Jméno, Číslo)

// vlastnosti
(     attribute              String      Jméno;
      attribute              String      Číslo;
      relationship           List        <Část>  má_části
            inverse          Část::je_částí
                  {order_by Část::Číslo};
      relationship           Set <Kurz> má_prerekvizity
            inverse          Kurz::je_prerekvizitou;
      relatioship            Set<Kurz> je_prerekvizitou
            inverse          Kurz::má_prerekvizity;

// operace
      Boolean Nabídka (in Integer semester) raises (již_nabízen);
      Boolean Zrušení (in Integer semester) raises (není_nabízen);
}
```




```
interface  Část
(     keys        (je_částí, Číslo))

{     attribute              String      Číslo;
      relationship           Profesor je_vyučován
            inverse          Profesor::vyučuje;
      relationship           Asistent má_asistenta
            inverse          Asistent::asistuje;
      relationship           Kurz je_částí
```

```
            inverse          Kurz::má_části
        relationship         Set <Student> je_zapsán
            inverse          Student::zapsal;

}

interface  Zaměstnanec
(       key (Jméno,Id))
{
        attribute            String      Jméno;
        attribute            Short       Id;
        attribute            Unsigned Int     roční_plat;
        void                 najmout();
        void                 vyhodit() raises (není_vhodný);
}
interface  Profesor: Zaměstnanec ()
{
        attribute        Enum Typ {řádný,docent, odb_asistent} typ;
        relatioship          Set <Část> vyučuje
            inverse          Část::je_vyučován;
        Short definitiva() raises (nevyhovuje);

}
interface  Asistent: Zaměstnanec, Student ()
{
        relationship         Část asistuje
            inverse          Část::má_asistenta;

}
interface  Student
(       keys Jméno,Student_id)
{
        attribute            String      Jméno;
        attribute            String      Student_id;
        attribute            Struct      Adresa
            {String koleje, String číslo_pokoje} p_adresa;
        relationship         Set<Část> zapsal
            inverse          Část::je_zapsán;
        Boolean registrace_kursu (in Unsigned Short Kurz,
            in Unsigned Short Část)
            raises (nesplněné_prerekvizity,obsazený_kurz,
                obsazená_část)
}
```

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 (chybovou podmínku), 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, **jedna_k_jedné** a **jedna_k_n**. Z důvodu kontroly konzistence je možné ke každému vztahu definovat jeho inverzní obraz. Nad vztahem je možné popsat i uspořádání.
- model obsahuje několik generátorů typů **collection**. Jsou to **set**, **bag**, **list** a **array**. Těmito generátory je možné organizovat vztahy typu jedna k_n. **Set** je klasická množina a **bag** multimnožina. **List** je uspořádaný seznam s možností přímého přístupu a **array** dynamické pole.

Model ODMG není tedy obvyklým rozšířením relačního modelu. Vychází z objektové implementace E-R modelu včetně implementace vztahů. Ovládacím jazykem je C++ resp. Smalltalk a v části C++ je návrh normy dotažen až do fáze implementace.

Elegantní je organizace dotazů. Dotaz k databázi je integrální součástí výrazu dotazovacího jazyka OQL, což umožňuje jednotně definovat popis konstruktorů typů, výběr z databáze a standardní operace výrazu do jednotného celku. Vlastnosti dotazovacího jazyka lze shrnout do následujících bodů:

- OQL umožňuje v rámci dotazu volat metody a naopak metody psané v ovládacím jazyce mohou zahrnovat dotazy,
- OQL poskytuje deklarativní přístup k údajům,
- OQL poskytuje prostředky vyšší úrovně pro práci s množinami objektů,
- OQL může být dobře optimalizováno vzhledem k jeho deklarativnímu základu.

V části konstrukce dotazů je sloučena jak sémantika klasického výrazu, tak reprezentace konstruktorů objektů spolu s vlastními dotazy. Uvažujeme-li předchozí příklad, je možné vytvořit objekt třídy zaměstnanec zápisem

```
Zaměstnanec (Jméno: „Petr", Id:234, roční_plat:120000)
```

V části vlastních dotazů lze nalézt operátor univerzální kvantifikace. Následující dotaz vrací `true`, pokud všichni studenti mají kladnou hodnotu jejich studentské identifikace.

```
for all x in Studenti: x.Student_id > 0
```

Následující operátor existenční kvantifikace vrací `true`, pokud existuje student, který si zapsal kurz vyučovaný profesorem Turingem.

```
exists x in Studenti.zapsal:x.vyučuje=„Turing"
```

Operátor select vytváří kolekci. V následujícím příkladu vznikne multimnožina dvojic studentů a jejich vyučujících profesorů, ovšem pouze těch s typem „řádný".

```
select dvojice (student:x.Jméno,prof:y.je_vyučován.Jméno)
     from x in Studenti,
          y in x.zapsal
     where y.je_vyučován.typ=„řádný";
```

Na závěr uveďme sémanticky poměrně složitý dotaz **group**. Následující příkaz vytvoří množinu ze tří struktur. Každá z nich bude obsahovat položku nazvanou **partition**, která bude opět množinou těch Zaměstnanců, kteří vyhovují zadanému predikátu. Kromě toho obsahuje každá struktura bude obsahovat položky nízký, střední a vysoký s hodnotami příslušných výrazů.

```
group x in Zaměstnanci
      by(nízký:  x.roční_plat < 100000,
      střední:   x.roční_plat >=100000
            and x.roční_plat< 200000,
      vysoký:    x.roční_plat >=200000)
```

Výsledným typem předchozího příkladu bude tedy

```
set<struct<nízký:boolean,střední:boolean,vysoký:boolean,
      partition: set <Zaměstnanci>)>
```

Pokud bychom chtěli platy rozdělit po 10000 úsecích, mohli bychom zadat

```
group x in Zaměstnanci
      by (úsek:  x.roční_plat div 10000)
```

4. Vývoj objektově orientovaného databázového systému

Pracovníci Ústavu informatiky a výpočetní techniky FEI VUT v Brně se v současné době zabývají vývojem objektově orientovaného databázového prostředí. V situaci volby moderního implementačního prostředku pro vývoj databázových aplikací je v současnosti firma VEMA Brno (počítače a programování), která provedla dosud okolo 1500 instalací informačního systému EKOS založeného na relačním modelu. Vzhledem k tomu, že v současnosti jsou výzkumné plány obou organizací blízké, spolupracují při vytváření vhodného databázového systému, který by umožňoval efektivní implementaci informačních systémů. Ze dvou možných cest modernizace (zvolit a použít některé ze současně dodávaných databázových prostředí, resp. vyvíjet vlastní produkt) byl nakonec zvolen druhý přístup.

Cílem projektu je vyvoření objektově orientovaného databázového systému. Vývoj systému je rozdělen do několika fází. V první fázi jde o návrh koncepce systému a zejména o návrh databázového modelu. Současně probíhá implementace prototypu. Další fáze představují převod prototypu na reálný provoz, návrh a implementaci aplikací.

Celý projekt je rozdělen do následujících oblastí:

- **Základní programátorské prostředí**. Tvoří nejnižší vrstvu projektu, která poskytuje základní abstrakce pro programování v implementačním jazyce C++. Zahrnuje veškerou vazbu na hostitelský operační systém. Definuje rovněž normy programování v celém projektu.
- **Databázový stroj**. Realizuje vnitřní uložení objektového modelu. Řeší problematiku fyzického provozování databáze a databázový server.
- **Řízení projektů**. Zahrnuje distribuované zpracování aplikací, problém přístupu k aplikacím a projektovou integraci.
- **Interpret metod**. Definuje vnitřní abstraktní kód zásobníkového charakteru, jeho interpret a ladicí prostředky. Řeší rovněž problém vnitřního tvaru jednoduchých datových typů.

- **Definiční a programovací jazyk**. Jazykové prostředky pro popis uživatelských aplikací, jejich kompilace a definice katalogu dat.
- **Správce objektů**. Představuje základní komunikační prostředí uživatele. Realizuje výběr projektu, objektů a spouštění metod.
- **Grafické rozhraní**. Řízení dialogu aplikace s uživatelem a dynamický návrh dialogu.
- **Systémové aplikace**. Zahrnuje různé samostatné systémové aplikace.

Z hlediska návaznosti řešení jednotlivých oblastí je klíčovým bodem návrh modelu, který se bezprostředně odráží v návrhu definičního a programovacího jazyka. Koncepce modelu vychází z návrhu normy ODMG. Je však k dispozici vlastní manipulační jazyk, který je navržen konzistentně s jazykem pro definici dat, čímž je zcela vyloučena impedance.

V současnosti je navržena základní verze jazyka pro popis dat a manipulaci s nimi a probíhá implementace překladače. Rovněž je navržen vnitřní abstraktní kód pro reprezentaci metod.

Literatura

[1] *Jacobson,I.*: Object-Oriented Software Engineering - A Use case Driven Approach, Addison Wesley, ACM Press 1992, p.524

[2] *Ellis,M.,A., Stroustrup,B.*: The Annotated C++ Reference Manual, Addison - Wesley Publishing Company 1992, p.453

[3] *Hudhes,J.,G.*: Object-Oriented Databases, Prentice Hall 1991, p. 280

[4] *Gray,M.D.P., Kulkarni, K.G., Paton,N.W.*: Object-Oriented Databases - A Semantic Data Model Approach, Prentice Hall 1992, p. 237

[5] *Catell, R.G.G.*: The Object Database Standard: ODMG - 93, Release 1.1, Morgan Kaufmann Publishers 1994 , San Francisco, p. 176

[6] *Máčel,M.*: Objektově orientované databáze, in: Sborník příspěvků 5. mezinárodního semináře Počítačové sítě a informační systémy 94, PC DIR Brno 1994, pp. 157 - 165

[7] *Beneš,M., Hruška,T., Máčel,M.*: Objektově orientované funkcionální modelování, in: Sborník přednášek XVI. mezinárodního kolokvia Vybrané problémy simulačních modelů 1994 - Katedra informatiky FEI VŠB Ostrava 1994, pp. 46-54

[8] *Hruška,T.*: Objektově orientované databáze a jejich použití pro realizaci informačních systémů, in: Sborník přednášek mezinárodního semináře EIS 95 - Zlín 1995 , VUT fakulta technologická pp.31-37

[9] *Hruška,T.*: DAPLEX - funkcionální objektově orientovaný model, in: Sborník přednášek XVII. mezinárodního kolokvia Vybrané problémy simulačních modelů 1995 - Katedra informatiky FEI VŠB Ostrava 1995, printed

[10] *Pokorný, J.*: Objektově orientovaný přístup - jediná varianta? Sborník semináře DATASEM'92, 1992, str.5-16.

[11] Next-Generation Database Systems. Special section. Communications of the ACM, Vol.34, No.10, October 1991.

[12] *Coad,P., Yourdon,E.*: Object-Oriented Analysis, Yourdon Press, Engelwood Cliffs, New Jersey 1991

[13] *Wilkinson K., Lyngbaek P., Hasan W.*: The Iris Architecture and Implementation. IEEE Transactions on knowledge and data engineering. Vol. 2, No. 1, March 1990.

[14] *Kuklová,J.,Popelínský,L.*: Our Experience with Public-Domain Object-Oriented Databases, Sofsem'93 Conference Proceedings, MU Brno 1993, str.59-62

[15] *Merunka,V., Polák, J.*: Objektově orientovaná analýza a design, In: Sborník semináře Programování 94, Dům techniky Ostrava 1994, str. 17-34

[16] *Molhanec,M.*: Některé souvislosti mezi převodem objektově orientovaného modelu do modelu entitně-relačního, In: Sborník semináře Programování 94, Dům techniky Ostrava 1994, str. 47-50

[17] *Pokorný, J.*: Objektově orientované databáze, In: Sborník semináře Programování 93, Dům techniky Ostrava 1993, str. 80-91

Autor

Doc. Ing. Tomáš Hruška, CSc.,
Ústav informatiky a výpočetní techniky FEI VUT
Božetěchova 2
612 66 Brno
tel.:05/7275 238, fax:05/4121 1141
e-mail: hruska@dcse.fee.vutbr.cz