VÝUKA OBJEKTOVÉHO MODELOVÁNÍ POMOCÍ SMALLTALKU

Vojtěch Merunka, Athanasios Podaras
Katedra informačního inženýrství, PEF, ČZU Praha
merunka@pef.czu.cz, podaras@pef.czu.cz

ABSTRAKT:
Příspěvek se zabývá otázkami nového způsobu výuky objektového modelování. Dále seznamuje s programovacím jazykem Smalltalk jako vhodným nástrojem pro praktickou demonstraci principů objektového návrhu a objektových databází. Součástí příspěvku je stručný popis aplikace Daskalos, který autor naprogramoval jako nadstavbu k prostředí VisualWorks for Smalltalk.

KLÍČOVÁ SLOVA:
Smalltalk, VisualWorks, Daskalos, objektové modelování, objektové databáze, objektově orientovaná analýza, UML

1 Úvod – revoluční versus evoluční přístup k výuce

V dnešní době již není třeba diskutovat o tom, že objektově orientovaný přístup je důležitou součástí výuky informatiky. Jedná se o problematiku, která přesahuje rámec jednoho vyučovaného předmětu. Dohromady se objektový přístup týká více než poloviny všech vyučovaných informatických předmětů.

Odborníci jako např. E. Yourdon, B. Meyer a další se shodují v názoru, že čím dříve se začne s výukou OOP, tím lépe. Autor tohoto článku má dobré praktické zkušenosti s výukou čistého objektově orientovaného programování (Smalltalk od roku 1991) a nerelačních objektových databází (Gemstone od roku 2002), které potvrzují tyto názory. OOP je samo o sobě syntézou disciplín, které kladou přiměřeně vysoké nároky na abstraktní myšlení a jako takové představují pro výuku velmi vhodnou intelektuální disciplínu. Nevidíme žádný důvod pro otálení a předkládání OOP jako vyvrcholení řetězu výuky, který začínal „klasickými“ předměty. Podle našich vlastních zkušeností výuku OOP s nadšením přijímají především lepší studenti a to nejen proto, že se jedná o moderní trend, ale také proto, že nástroje OOP, jsou-li vhodně vybrány, jsou srozumitelné, efektivní a stimulují abstraktní a logické myšlení. Varianta, kdy se informatika zahájí výukou OOP je sice netradiční, ale má dva z pedagogického hlediska významné přínosy:

1. Nedochází zde k efektu, kdy jsou nejprve vyloženy „tradiční“ metody a postupy tvorby software, od kterých se však později musí student oprostit, když je konečně zahájen kurz OOP.
2. Zahájení výuky informatiky pomocí soudobé v praxi používané technologie (tedy prostředků OOP) a učení dalších přístupů později je v souladu se záměrem členit výuku na úvodní praktickou „bakalářskou“ etapu a následnou více teoretickou „inženýrskou“ etapu studia jdoucí dále do hloubky „computer science“, což odpovídá systémům studia na většině univerzit ve světě.

Právě skutečnost, že zde hovoříme o bakalářském stupni studia, považujeme za nejdůležitější. Je třeba si totiž uvědomit, že značná část studentů nebude pokračovat do magisterského stupně studia a půjde rovnou do praxe. Proto jim potřebujeme podat ucelenou sadu použitelných znalostí. Výuka historie výpočetní techniky, byť jakkoliv fundovaná, je bakalářům absolventům skoro k ničemu.

Takoví studenti potřebují jiné znalosti, které samozřejmě přesahují středoškolskou úroveň, ale zároveň je nezajímá všechno z toho, co je určeno pro posluchače magisterského stupně.

Nehovoříme tu ale o podávání kojenecké stravy po lžičkách. Zahájení výuky programování objektovým paradigmatem není „jednodušší“. Jestliže se totiž má učit OOP jako první metodologie tvorby software, potom také musí připravovat půdu pro pozdější výuku dalších souvisejících disciplín, jako například funkcionální či logické programování, databázová technologie, imperativní jazyky, projektování informačních systémů apod. Tato podmínka je ale dobře splnitelná a v praxi jsou známy příklady, kdy předchozí kurz OOP měl blahodárný vliv například na budoucí programátory v C či Cobolu, neboť seznámení se s abstraktnějším a logičtějším programovacím paradigmatem následně vede k čistšímu a efektivnějšímu využívání „klasičtějších“ prostředků.

Do značné míry je tato diskuse opakováním historie výuky výpočetní techniky. Před cca 20 lety totiž proběhla malá revoluce ve výuce programování. Dříve se zahajovalo počítačovým hardwarem a strojovými jazyky a vyšší programovací jazyky byly vyučovány „až potom“. Mnozí z nás si ještě pamatují změnu ve prospěch strukturovaného programování ve vyšším programovacím jazyku Pascal jako prvního programovacího paradigmatu. Domníváme se, že diskutovaný přechod na objektové paradigma zde má svoji analogii.

2 Výuka objektového modelování jako začátek řetězu výuky programování

Již i u nás jsou pokusy zahájit výuku programování kurzem objektového modelování nebo programování. Je tomu tak například na Univerzitě Hradec Králové a v přípravě jsou i nové předměty na FEL ČVUT v Praze. Praxe většiny českých vysokých škol je ale následující:

1. Na začátku studijního plánu se vyučují základy programování pomocí jazyka Java, jako projev moderního trendu, který smetl Pascal do propadliště dějin.
2. Později v průběhu studia se studenti seznamují v předmětech softwarového inženýrství s různými metodikami tvorby softwaru v jejichž diagramech se operuje s pojmy OOP.
3. Čisté OOP v programování a nebo v databázových systémech je odloženo až na konec magisterského studia v různých vesměs volitelných předmětech. Většinou se ale po celou dobu studia studenti seznamují jen s hybridními objektovými programovacími jazyky (Java nebo C++) a relačně objektovými databázemi (Oracle). S čistými objektovými jazyky a nerelačními objektovými databázemi se nesetkají.
4. Teoretické základy OOP a jeho souvislosti s teorií grafů, lambda kalkulem a formálními technikami návrhu datových struktur jsou opomíjeny. Výuka je zaměřena jen na „psaní kódu“.

Většinou se používá jazyk Java. Objektové programování se v Javě ale často vykládá jen jako jakýsi nezbytný doplněk k syntaxi tohoto jazyka. Jenže smyslem objektového návrhu není umět psát prográmky, které používají (samozřejmě objektové) knihovny pro různé barevné efekty na obrazovce a nebo pro ovládání nejrůznějších (samozřejmě objektových) komponent z knihovny. V takových programech totiž při bližším zkoumání nacházíme vesměs jen objekty, jejichž atributy i parametry zpráv jsou pouze skalární hodnoty typu znak, číslo, grafická souřadnice, ….

To, co skutečně potřebujeme, je umět navrhovat DATOVÉ objekty pro softwarové aplikace. To znamená tvořit vlastní třídy, instance a kolekce objektů, jejich atributy jsou další objekty, a které vytvářejí pomocí vzájemných vazeb dědění a skládání netriviální soustavu objektů modelující nějakou praktickou úlohu. Takový „program“ nemusí být ani animovaný ani jinak grafický a přesto je na rozdíl od jiných důsledně objektový.

3 Jak by měl vypadat úvodní kurz objektového modelování?

Požadavky na úvodní předmět lze shrnout do následujících bodů:

1. Zahájit přednáškami o teoretických základech programování a vyložit lambda kalkul. Ten potřebujeme proto, že jde o ideální prostředek pro pozdější výklad chování objektů.
2. Pokračovat základy OOP. Zde je vyložen pojem třída, instance a kolekce objektů, dědičnost objektů, skládání objektů a polymorfismus objektů. Lambda výrazy se zde objeví nejen v podobě kódu objektových metod, ale také jako parametry zpráv i jako samostatný objekt.
3. Výklad vybraných technik návrhu jako je objektová normalizace a aplikace vybraných návrhových vzorů.
4. Základy datových manipulací s objekty a s kolekcemi objektů. Jde o dotazování v duchu databázových systémů a o problematiku změny struktury schématu a migrace objektů mezi různými verzemi.
5. Při výuce využívat nejen vlastní programovací jazyk, ale i vizuální modelovací software, ve kterém bude možné s objekty přímo pracovat. Zde se formou praktických příkladů studenti seznámí se základy notace diagramu tříd UML.

Na tento základ může být navazováno výukou projektování informačních systémů i databázových systémů. Studenti se totiž již v úvodním předmětu prakticky seznámili s modelováním a naučili se používat zjednodušený diagram tříd UML. Výuku databází bude možné začít výkladem všech dnes používaných tří datových modelů – síťového, relačního a objektového. Každý z těchto datových modelů může být vykládán jako konkrétní implementační varianta toho, s čím se už dříve studenti seznámili.

4 Smalltalk jako vhodný úvodní programovací jazyk

4.1 Stručná historie Smalltalku

Smalltalk byl vyvíjen v Kalifornii v Palo Alto Research Center (PARC) kolektivem vědců vedených dr. Alanem Kayem (tým Learning Research Group) a dr. Adelou Goldbergovou (tým System Concepts Laboratory) v létech 1970-1980. Předmětem celého výzkumu, který byl financován v největší míře firmou Xerox, byl projekt "Dynabook" pro vývoj osobního počítače budoucnosti.

Počítač Dynabook se měl skládat z grafického displeje s jemnou bitovou grafikou, klávesnicí, v té době novou periférií - perem, později nahrazeným myší - a jeho součástí měl být i síťový interface. Pro vzhled systému byla poprvé na světě použita překryvná okna, vynořovací menu a ikony. V průběhu 70. let bylo dokonce vyrobeno několik prototypů takových počítačů. Předpokládalo se, že počítač bude obsahovat jednotné softwarové prostředí, které bude současně plnit úlohu operačního systému i programovacího jazyka s vývojovými nástroji. Právě tento software dostal název Smalltalk.

Ve Smalltalku, který byl jako projekt dokončen v roce 1980, se nejvíce odrazily prvky z jazyka LISP a z prvního objektově orientovaného jazyka Simula. Část týmu v PARC zůstala a založila pod vedením A. Goldbergové firmu ParcPlace Systems (dnes Cincom), která rozvíjí Smalltalk dodnes, jiní spolu s A. Kayem odešli do firmy Apple Computer, kde poté uvedli na trh první dostupný komerční osobní počítač Lisa s grafickým uživ. rozhraním (dále jen GUI). Smalltalk a jeho GUI byl v průběhu 80. let využíván zpočátku pouze na výkonných pracovních stanicích té doby, z nichž nejznámější byl Tektronix 4404 z roku 1982. V USA vzniklo několik firem (např. Knowledge Systems), které již okolo roku 1985 používají Smalltalk pro náročné aplikace z oblasti expertních

systémů, řízení výroby, projektového řízení apod. Dnes se Smalltalk prakticky využívá nejvíc v USA. Jsou to např. informační systémy firem Navigant International Northwest Travel, Florida Power & Light, Orient Overseas Container Line nebo software pro analytickou firmu JP Morgan pracující pro finančníky na Wallstreetu. Smalltalk je používán také ve výzkumu na vysokých školách. Myšlenka GUI Smalltalku dala během 80. let vznik systémům Macintosh OS, MS Windows, X-Window apod. Smalltalk přímo ovlivnil vznik programovacích jazyků Objective-C, Actor, Eiffel, CLOS, Object Pascal, C++, Self, Ruby, Python, Oberon, Java a OCL ze standardu UML. Smalltalk je také jazykem objektových databází, z nichž je v praxi nejrozšířenější Gemstone.

Kromě komerčně využívaného prostředí firmy Cincom VisualWorks existují ještě dvě velmi kvalitní freewarové implementace: Smalltalk-X, který je zajímavý svou originálním způsobem vytvořenou vazbou na jazyk C s možnostmi překladu do strojového kódu. Je dostupný na jakémkoliv počítači s operačním systémem UNIX. Autor tohoto textu se v letech 1992-96 podílel na jeho vývoji a testování. Druhou implementací je systém Squeak. Je to rozšířený a podporovaný systém především na amerických univerzitách. Squeak je totiž znovuvzkříšený původní Dynabook. Jde o nejčistší verzi Smalltalku a je také zajímavý výzkumem v oblasti nových uživatelských rozhraní pro distribuované objekty (projekt Croquet mající trojrozměrné grafické uživatelské rozhraní).

4.2 Ukázky jazyka

Smalltalk je integrován s programovacím prostředím, které je napsané taktéž v jazyce Smalltalk. Vše je přístupné včetně zdrojových kódů (a to i v komerčních implementacích). Navenek se systém chová jako jediný rozsáhlý program nebo ještě lépe jako operační systém, který je programátorem používán a měněn za svého chodu. I když Smalltalk podléhá vývoji v oblasti OOP, tak zde popsané vlastnosti jsou jeho součástí již od roku 1976.

Důležitou součástí jazyka Smalltalk jsou bloky výrazů. Bloky výrazů jsou implementací lambda kalkulu. Blok je ve Smalltalku objektem, může být pojmenován, mohou mu být posílány příslušné zprávy, a může být použit v jiných výrazech (zprávách) jako příjemce nebo jako parametr. (Podle stejného principu jsou ve Smalltalku implementovány i metody objektů.) Výrazy v blocích se vyhodnocují vždy až při příslušném požadavku na jejich vyhodnocení, a ne při vytvoření bloku. Tentýž blok proto může v různých situacích vracet různé výsledky. Získaná hodnota samozřejmě záleží na stavu systému v době spuštění bloku a ne na stavu v době vytvoření bloku. Následující příklad ukazuje blok kódu, který je uschován do objektu se jménem `B`. Blok obsahuje kód, který umocňuje vstupní parametr na druhou a přičítá k tomuto výsledku hodnotu objektu `A`:

```
B := [:x | (x ** 2) + A].
```

Jestliže budeme mít v systému takto vytvořený blok, tak můžeme také nastavit objekt `A` na jinou hodnotu, než měl v době vytvoření bloku, a blok postupně spouštět jako například:

```
A := 10.
B value: 3.
```

nám dá výsledek 19 ($3^2 + 10$) nebo

```
A := 5.
B value: 4.
```

nám dá hodnotu 21 ($4^2 + 5$).

Další ukázkou zajímavostí jazyka Smalltalk je zpráva `perform:`. Je to zpráva, která posílá svému příjemci další zprávu podle svého parametru. Ve Smalltalku je totiž i zpráva považována za objekt,

který lze také přiřadit do proměnné. To si ukážeme na jednoduchém příkladě aritmetických operací. (Aritmetické operace jsou implementovány také pomocí zpráv posílaných číselným objektům):

```
20 sin.          nám dá hodnotu sin(20) nebo např.
20 cos.          nám dá hodnotu cos(20).
```

Použijeme-li zprávu `perform:`, tak můžeme selektor zprávy pro sinus nebo cosinus uložit jako proměnnou (v ukázce to je objekt se jménem `C`) a aritmetickou operaci vyvolat následovně:

```
C := #sin.          a potom výraz
20 perform: C.      nám dá opět hodnotu sin(20).
```

Díky bohaté knihovně Smalltalku (typická instalace VisualWorks má asi 8000 tříd a metatříd s přibližně 70.000 různými selektory zpráv v 5.500.000 metodách) lze Smalltalk používat i jako databázový dotazovací jazyk, jak naznačují následující ukázky:

```
#(2 3 6 5 4 7 8) select: [:x | x > 5].
```

Dává výsledek `#(6 7 8)`, protože vybere z pole prvky, které jsou větší než `5`.
Lambda výraz `[:x | x > 5]` v parametru zprávy `select:` slouží k vymezení podmínky selekce.

```
Osoby select: [:x | x vek > 18].
```

Vybere z množiny `Osoby` ty osoby, které jsou starší `18` let.

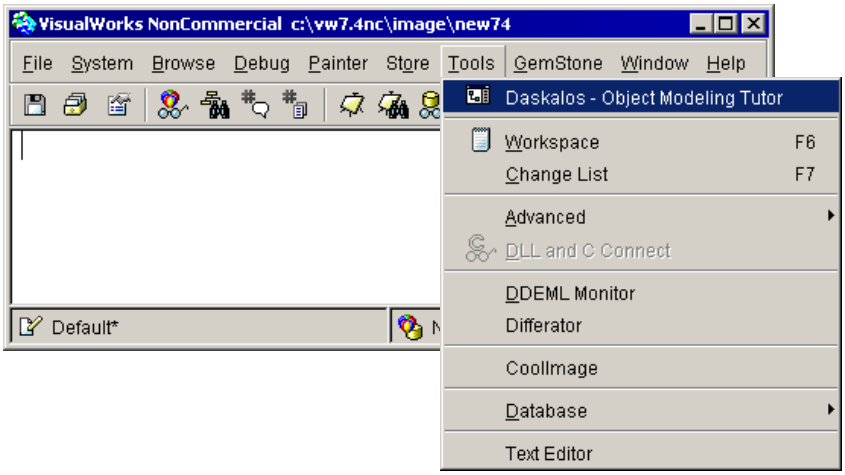
A úplně na závěr ukázka složitějšího dotazu, který vybírá jména zákazníků a data kontraktů pro kontrakty, které byly uzavřeny na výrobky od výrobců z Prahy. Pro větší porozumění uvedeme tento dotaz i v jazyce OQL, který je variantou SQL pro objektové databáze:

```
(Contracts select: [:c | c product producer address = 'Prague'])
    collect:  [:c | c customer name]
       with:  [:c | c date].
```

```
SELECT  c.customer.name , c.date
  FROM  c  IN  Contracts
  WHERE c.product.producer.address = 'Prague';
```

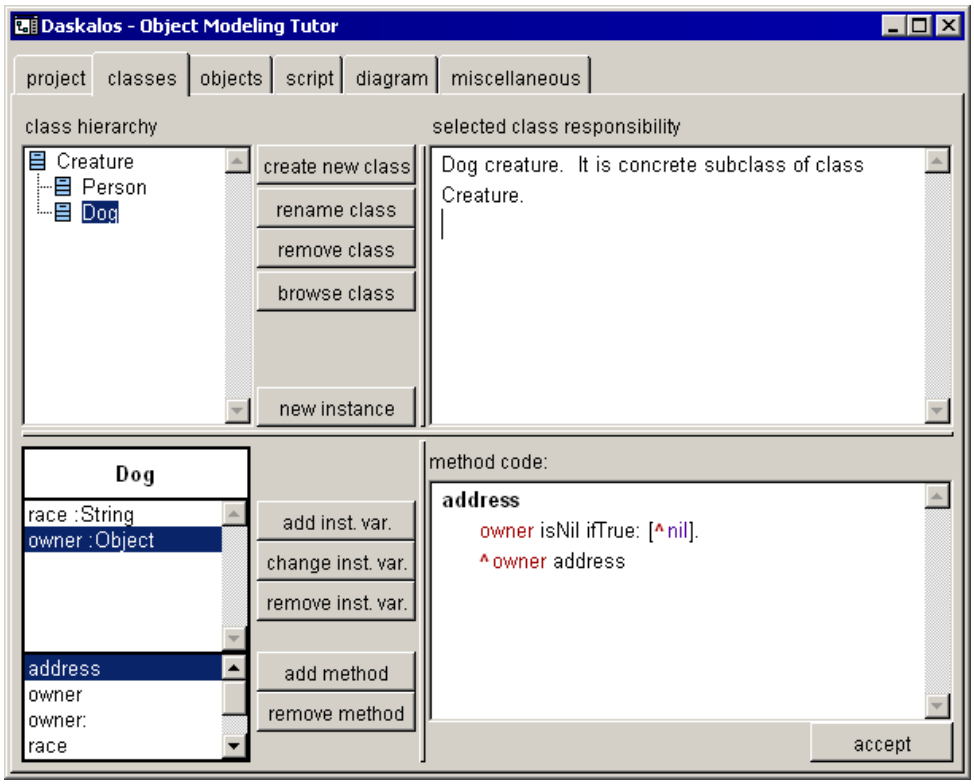
5 Daskalos

Daskalos je počítačový program, který slouží k výuce objektově orientovaného modelování podle zásad diskutovaných v tomto článku. Je naprogramován jako samostatná parcela systému VisualWorks/Smalltalk verze 7.4, která je pro účely výuky a výzkumu zdarma.



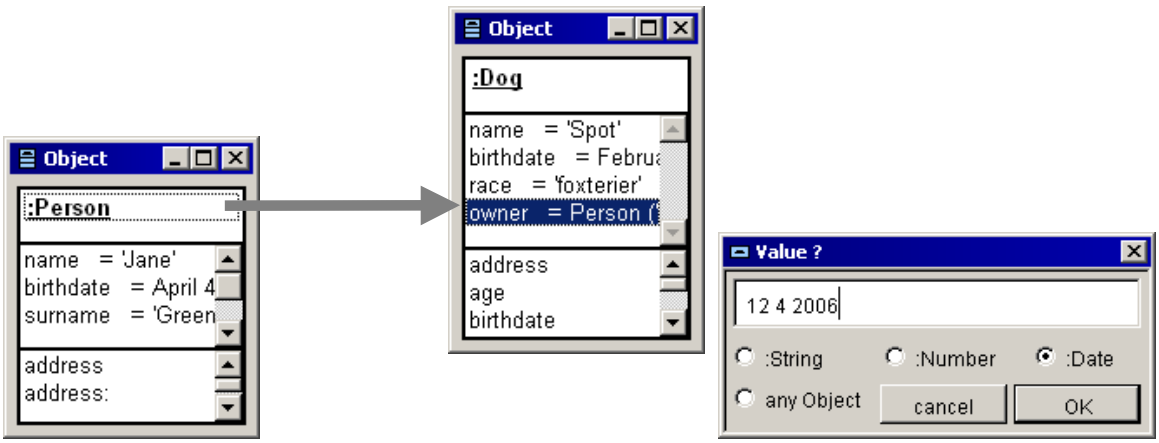
obr. 1. spuštění Daskalu z VisualWorks

V Daskalu je možné vytvořit třídy a množiny objektů a programovat metody. Protože vzniklý kód je součástí standardního vývojového prostředí Smalltalku, tak lze Daskalos použít jako vizuální nástroj pro tvorbu datových objektů určených pro běžně vyvíjené aplikace.



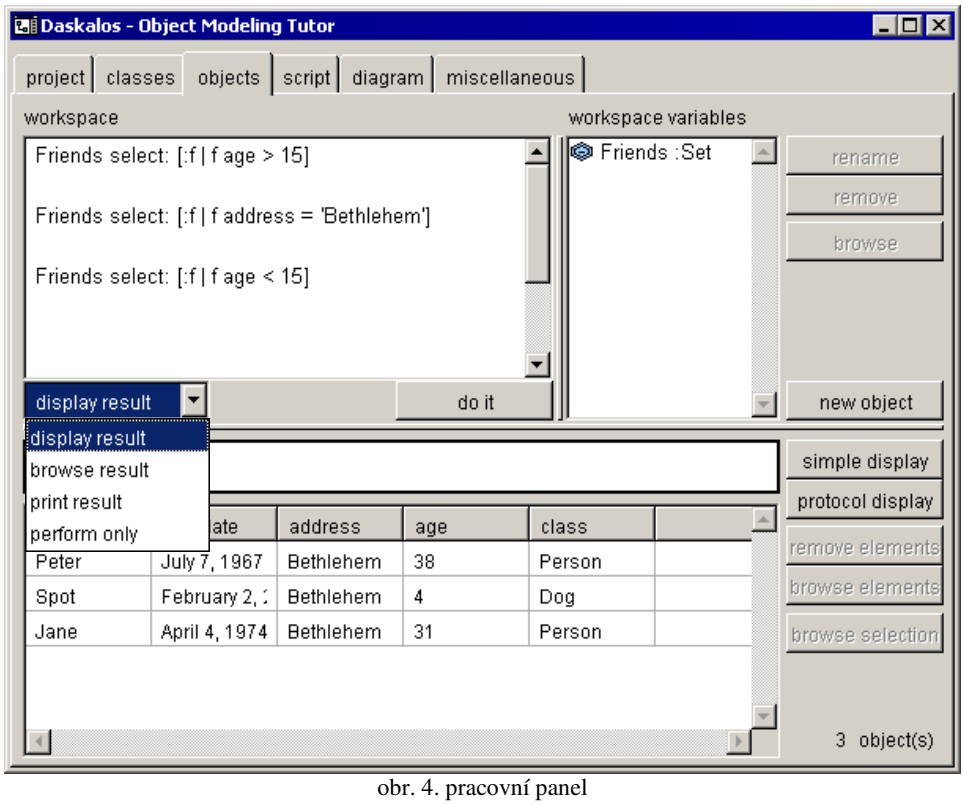
obr. 2. tvorba tříd a metod

Daskalos také dovoluje objekty testovat. Objekty a třídy objektů jsou zobrazovány podle standardu UML přičemž s obsahem takto zobrazených symbolů lze přímo pracovat. Objekty lze vyčleňovat do samostatných oken a takto zobrazeným objektům lze také posílat zprávy přímo kliknutím na zobrazený objekt nebo přímo manipulovat s atributy objektů způsobem táhni-pusť:



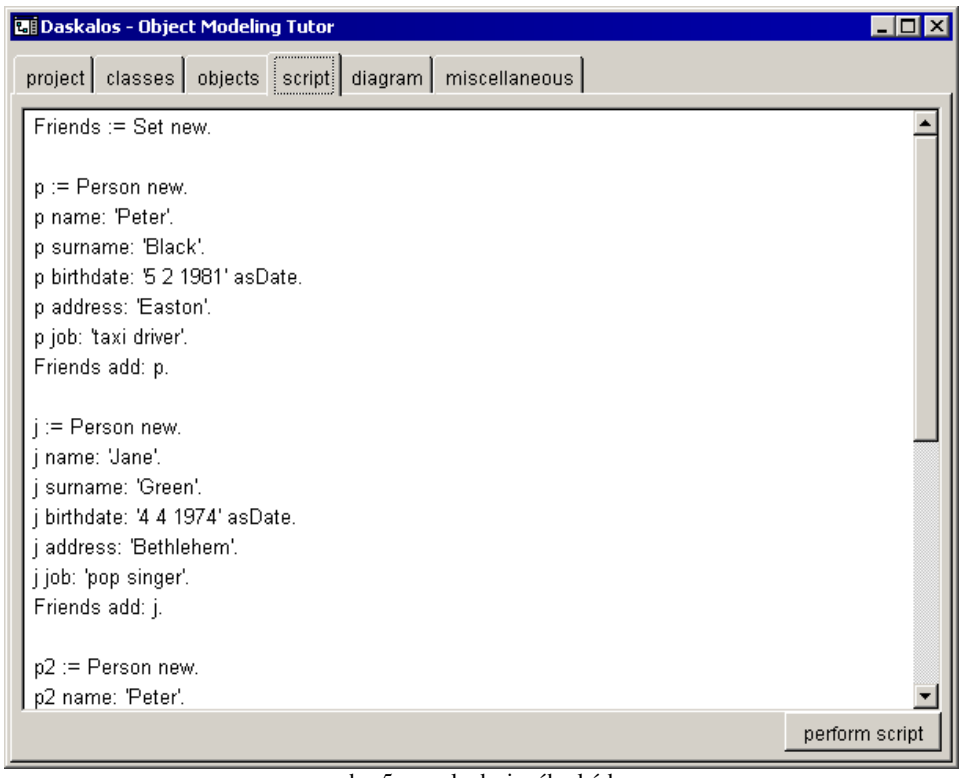
obr. 3. manipulace s objekty

Pro komplikovanější operace s objekty – například kladení dotazů nad množinami objektů – je možné využít pracovní panel, ve kterém lze příslušné výrazy vyhodnocovat a pracovat s jejich výsledky:



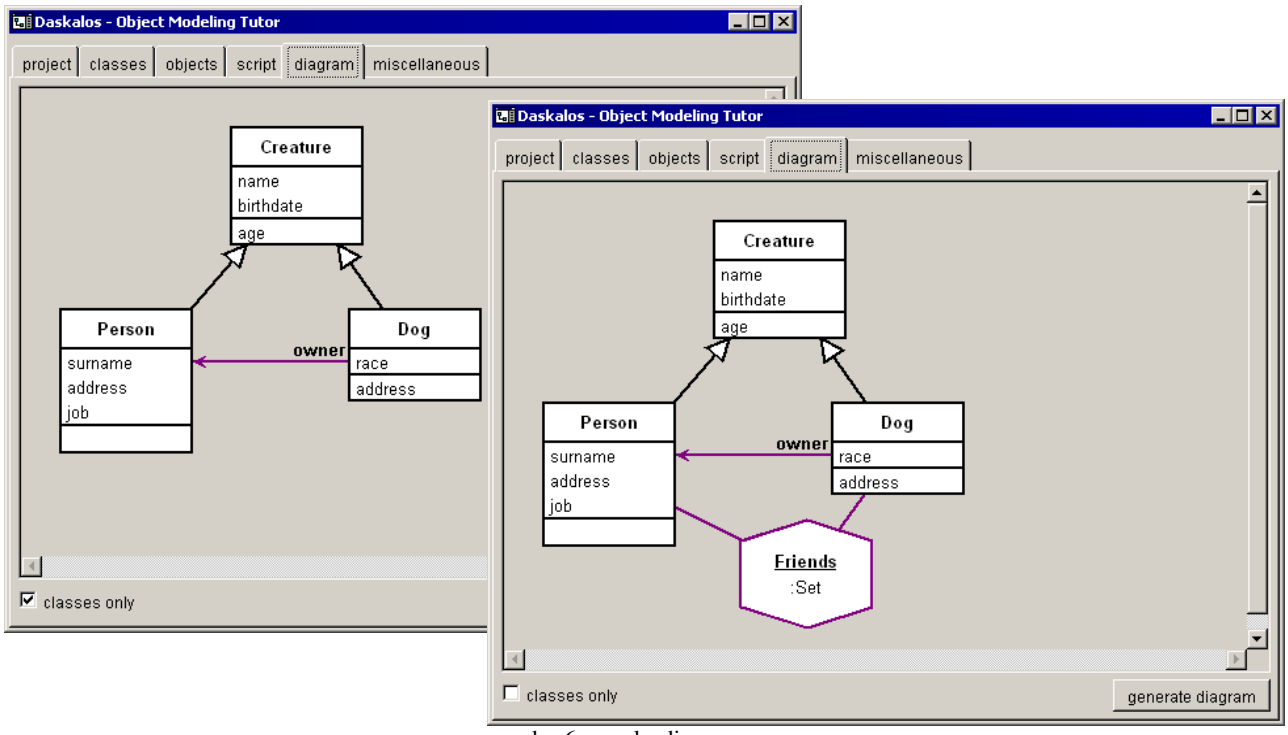
obr. 4. pracovní panel

Objekty, které jsou potřeba k testování, je možné vytvářet nejen vizuálními prostředky, ale i obvyklým způsobem ze zdrojového kódu.



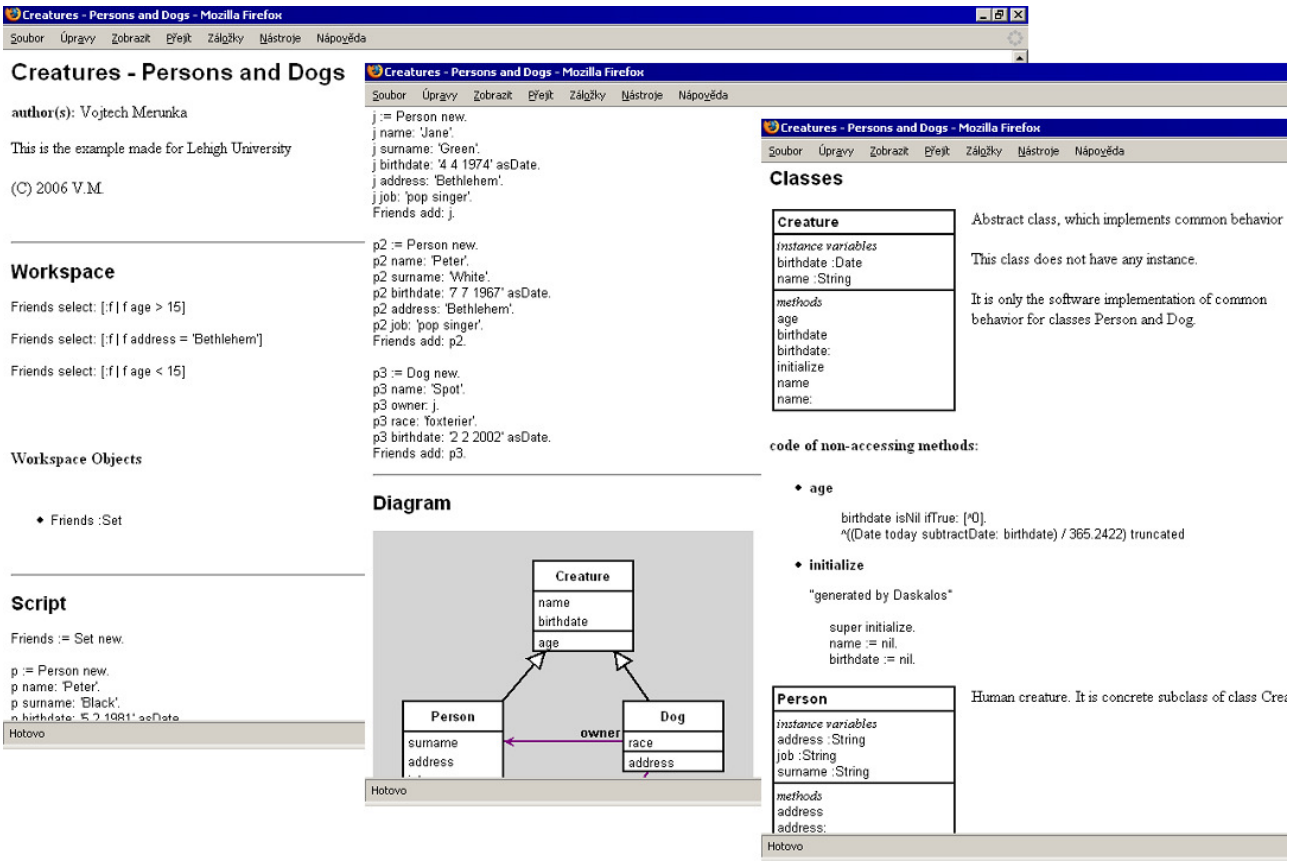
obr. 5. panel zdrojového kódu

Třídy a množiny objektů, se kterými se pracuje, jsou v Daskalu zobrazovány také v podobě diagramu tříd. Symboly tohoto diagramu, jejich obsah a vazby mezi nimi jsou synchronizovány se skutečným obsahem objektů z pracovního panelu. To znamená, že podoba diagramu se mění podle toho, jak se s objekty v pracovním panelu pracuje.



obr. 6. panel s diagramem

Daskalos je primárně určen pro výuku OOP začátečníkům. Proto se projekt ukládá nejen do datového souboru ve formátu XML, ale je také generována dokumentace obsahující zdrojové kódy, data i diagramy ve formátu HTML.



obr. 7. dokumentace ve formátu HTML zobrazená webovým prohlížečem

6 Závěr

V době psaní tohoto článku jsme měli již první kladné ohlasy z výuky na PEF ČZU a FEL ČVUT. Zajímavá je také zkušenost z přednášek informačního managementu pro kurz MBA. Zde popisované objektové modelování pomocí programu Daskalos bylo použito pro návrh a ověření datových struktur navrhovaných ve studentských projektech organizační změny za pomoci ICT.

V tomto článku diskutované myšlenky jsou také podporovány grantem MSM6046070904 na výzkum v oblasti znalostních databázových systémů.

Literatura

1. Carda A., Merunka V., Polák J.: *Umění systémového návrhu - objektově orientovaná tvorba informačních systémů pomocí původní metody BORM*. Grada, Praha 2003. ISBN 80-247-0424-2.
2. Lacko B.: *Vademekum objektově orientované technologie*, sborník konference Programování, Ostrava 1993.
3. Merunka V.: *Objektový databázový systém Gemstone*, sborník konference OBJEKTY 2003. Ostrava 2003. ISBN 80-248-0274-0
4. Merunka V.: *Současná objektově orientovaná vývojová prostředí založená na jazyce Smalltalk*, ve sborníku konference Tvorba softwaru 2000, Tanger Ostrava 2000, ISBN 80-85988-49-6
5. Meyer B.: Towards an Object-Oriented Curriculum, Object Currents, http://www.sigs.com/publications/docs/oc/9602/oc9602.c.meyer.html
6. Molhanec M.: *Kritika některých chápání objektově orientovaného paradigmatu*, Sborník 30. ročníku konference Tvorba softwaru, Ostrava 2004

7. Pícka Marek, Pergl Robert, Merunka Vojtěch: *Objektově orientovaná tvorba softwaru*, skripta ČZU, Česká zemědělská univezita, Praha 2004, ISBN 80-213-1159-2
8. sborníky konferencí ECOOP, European Conference on Object-Oriented Programming
9. sborníky konferencí OOPSLA - Conference on Object-Oriented Programming Systems, Languages and Applications
10. Virius M., Merunka V., *Unifikovaný modelovací jazyk UML I., II. a III.*, série tří článků, Chip Vogel Publishing Praha 2002, ISSN 1210-0684
11. webová stránka http://www.cincom.com, týkající se nástroje VisualWorks/Smalltalk.
12. webová stránka autora http://kii.pef.czu.cz/~merunka, kde je kód modulu Daskalos a materiály k výuce.
13. webová stránka mezinárodního sdružení Object Management Group, http://www.omg.org