FORMALIZACE TRANSFORMACÍ V OBJEKTOVĚ ORIENTOVANÝCH MODELECH

Oldřich Nouza
ČVUT v Praze, Fakulta jaderná a fyzikálně inženýrská
Břehová 7, 115 19 Praha 1
nouza@km1.fjfi.cvut.cz

ABSTRAKT:
Transformations of models are very often applied in the present object oriented analysis and design. The main aim of this paper is to describe a unified view on common types of object oriented model transformations. First, it is explained the purpose of transformations in object oriented models, basic types of the transformations, and division of them into categories according to the common attributes. Then, basic rules describing a unification of the transformations are shown and some examples of usage of these rules in practice are outlined. Finally, some types of transformations are uniformly described using a formal transformational calculus.

KLÍČOVÁ SLOVA:
formalizace, objektová normalizace, návrhové vzory, refaktoring, transformace, transformační kalkul

ÚVOD

Transformace objektově orientovaných modelů jsou v moderní objektově orientované analýze a návrhu informačních systémů zcela běžnou záležitostí. V případě systémů velkého rozsahu je nemožné na základě seznamu požadavků vytvořit model a podle něj provést implementaci požadovaných funkčností. Model je potřeba několikrát upravit a doplnit, jinými slovy – transformovat, a sice tak, abychom se každou transformací více přiblížili výslednému modelu.

TRANSFORMACE OBJEKTOVÝCH MODELŮ

Transformací objektově orientovaného modelu v nejobecnějším slova smyslu rozumíme proces, kdy na základě určitého modelu (zdrojového) vznikne model jiný (cílový).
V následujících odstavcích se omezíme na případy, kdy transformace nemá vliv na význam původního modelu, čili modelovaný systém z hlediska funkcionality zůstává po transformaci nezměněn.

KATEGORIZACE TRANSFORMACÍ OBJEKTOVÝCH MODELŮ

Transformace objektově orientovaných modelů lze rozdělit do dvou hlavních skupin:

- Transformace mezi typy modelů – Zdrojový model je jiného typu než model cílový. Tato transformace se provádí např. při přechodu mezi etapami objektového návrhu.
  Příklad: transformace modelu nezávislého na platformě na model pro specifickou platformu.
- Transformace v rámci typu modelu – Zdrojový model je stejného typu jako cílový. Tuto transformaci lze provést, potřebujeme-li zajistit, aby model splňoval určitá stanovená pravidla.
  Příklad: refaktoring, použití návrhového vzoru, objektová normalizace. Transformace spadající do této skupiny budou rozebrány níže na příkladech.

PRAVIDLA PRO TRANSFORMACI OBJEKTOVÝCH MODELŮ

Nejprve zavedeme značení transformace modelů. Uvažujme objektové modely M_S a M_T. Zápisem $M_S \rightarrow M_T$ budeme vyjadřovat skutečnost, že model M_S *lze transformovat* na model M_T.
Pro libovolné objektové modely M_S, M_M a M_T platí:

1. Jestliže $M_S \rightarrow M_T$, pak $M_T \rightarrow M_S$, čili transformace modelů je symetrická.
2. Jestliže $M_S \rightarrow M_M$ a $M_M \rightarrow M_T$, pak $M_S \rightarrow M_T$, jinými slovy transformace modelů je tranzitivní.

Symetrie transformace nám říká, že ke každé transformaci existuje inverzní transformace. Příklady:

- přidání třídy do modelu × odebrání třídy, která není v relaci s jinými třídami, z modelu
- přesunutí společného atributu do předka × přesunutí atributu do všech potomků

Tranzitivita transformace je užitečná, pokud jsou příliš velké rozdíly mezi zdrojovým modelem M_S a cílovým modelem M_T, a přímá transformace $M_S \rightarrow M_T$ je tím pádem složitá. V tomto případě je vhodné transformaci rozložit na vhodnou posloupnost dílčích neboli *elementárních transformací*:

$$M_S = M_0 \rightarrow M_1 \rightarrow M_2 \rightarrow \ldots \rightarrow M_n = M_T,$$

Příklady rozkladu transformace na posloupnost elementárních transformací jsou uvedeny v následujících kapitolách.

PŘÍKLAD TRANSFORMACE MODELŮ – REFAKTORING

Pojem *refaktoring* lze chápat dvěma způsoby:

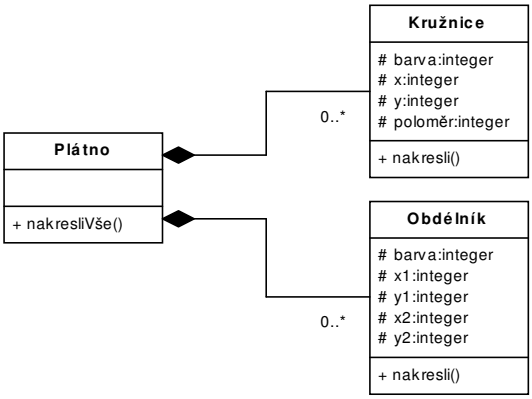
1. Refaktoring je libovolná změna struktury systému, přičemž chování systému zůstane nezměněno.
2. Refaktoring je taková změna v systému, která zvýší kvalitu struktury systému, ale nezmění chování systému.

Uvedené definice refaktoringu se nezmiňují o objektový modelech z toho důvodu, že refaktoring lze provádět jak na úrovni modelu (analýzy, návrhu systému), tak na úrovni kódu (implementace systému). Dále se omezíme pouze případy refaktoringu modelu.
První definice v podstatě odpovídá definici transformace modelu. V následujícím textu budeme uvažovat refaktoring ve smyslu druhé definice, která zahrnuje pouze transformace vedoucí ke kvalitativnímu zlepšení modelu, například:

- Zvýšení přehlednosti – Např. třída s příliš mnoha atributy ztrácí na přehlednosti. Vhodné rozdělení takovéto třídy na dvě či více přehlednost zvýší.
- Usnadnění dalších úprav – Toto přímo souvisí s přehledností. Přehlednější model lze rychleji rozluštit, a tím i upravit.

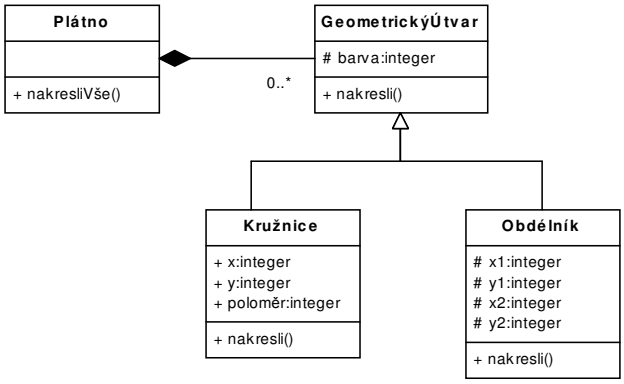
Z výše uvedených řádků vyplývá důležitost refaktoringu pro softwarový vývoj, ačkoliv z pohledu uživatelů je naprosto bezvýznamný.

Příklad 1. Na obr. 1 je znázorněn zjednodušený model tříd primitivního prohlížeče vektorových obrazců.



Obr. 1. Třídy Kružnice a Obdélník bez společného rozhraní

Třída Plátno reprezentuje plátno, na které se vykreslují obrazce, a zapouzdřuje třídy Kružnice a Obdélník. Třída Kružnice obsahuje atributy x, y a poloměr, které určují souřadnice středu a poloměr kružnice, a třída Obdélník atributy x1, y1, x2 a y2 vymezující příslušnou obdélníkovou oblast. Obě třídy mají navíc atribut barva představující barvu obrazce a metodu nakresli(), která zajišťuje o vykreslení daného útvaru. Třída Plátno obsahuje metodu nakresliVše(), jež vykreslí všechny zapouzdřené kružnice a obdélníky na plátno tím, že zavolá metodu nakresli() postupně všech zapouzdřených instancí tříd Kružnice a Obdélník.
Jelikož třídy Kružnice a Obdélník mají hodně společného, nabízí se provést refaktoring, který zavede jejich předka, abstraktní třídu GeometrickýÚtvar. Tato třída bude obsahovat společný atribut barva a abstraktní metodu nakresli(). Třída Plátno bude zapouzdřovat třídu GeometrickýÚtvar místo tříd Kružnice a Obdélník. Model po refaktoringu je zachycen na obr. 2.



Obr. 2. Třídy Kružnice a Obdélník implementující společné rozhraní GeometrickýÚtvar

Nyní si podrobněji popíšeme jednotlivé elementární transformační kroky:

1. Přidání nové abstraktní třídy GeometrickýÚtvar do modelu.
2. Přidání vazby dědičnosti mezi třídu GeometrickýÚtvar a každou ze tříd Kružnice a Obdélník (stanovení třídy GeometrickýÚtvar společným předkem tříd Kružnice a Obdélník).
3. Přesunutí společného atributu barva ze tříd Kružnice a Obdélník do třídy GeometrickýÚtvar.

4. Přidání společné metody nakresli() tříd Kružnice a Obdélník do třídy GeometrickýÚtvar jako abstraktní metody.
5. Sloučení agregačních vazeb mezi třídami Plátno a Kružnice a třídami Plátno a Obdélník do jediné agregační vazby mezi třídami Plátno a GeometrickýÚtvar. Tím dostaneme výsledek zachycený na obr. 2.

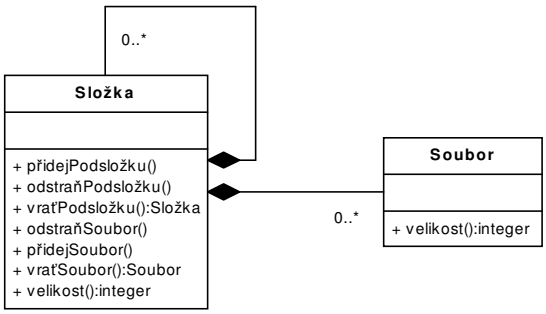
Poznámka. Bližší informace o refaktoringu modelů lze nalézt např. v [1].

PŘÍKLAD TRANSFORMACE MODELŮ – POUŽITÍ NÁVRHOVÉHO VZORU

Návrhovým vzorem rozumíme předpis sloužící k návrhům řešení stejného typu. Jinými slovy, návrhový vzor udává třídu návrhů řešení, zatímco konkrétní návrh řešení je instancí této třídy. Z toho plyne velká výhoda aplikace návrhových vzorů, a sice úspora času. Opakované použití návrhového vzoru na určité řešení je méně časově (a tím i finančně) náročné, než konstruovat návrh takového řešení od základů.
O návrhových vzorech a jejich klasifikaci je blíže pojednáváno v [2].
Aplikaci návrhového vzoru na řešení daného problému lze chápat dvěma způsoby:

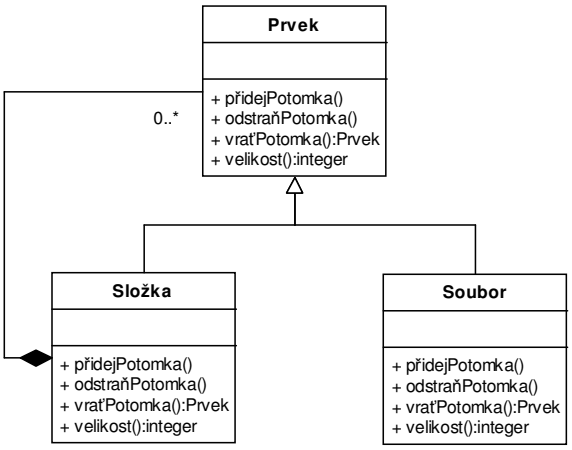
1. Na počátku máme prázdný model. V každém kroku přidáme do modelu jeden prvek (např. třídu, vazbu apod.), dokud se nedopracujeme výsledku, který není v rozporu s pravidly návrhového vzoru.
2. Vycházíme z modelu, který pravidlům návrhovému vzoru neodpovídá, ale je alternativním řešením daného problému. Jedná se tedy o jakýsi stavební kámen pro použití návrhového vzoru. Ve druhé fázi provedeme transformaci modelu do podoby, která již pravidlům návrhového vzoru odpovídá.

Příklad 2. Použití návrhového vzoru ve smyslu bodu 2 si ukážeme na příkladě.
Předpokládejme model struktury souborů a adresářů (obr. 3)



Obr. 3. – Model stromové struktury souborů a složek

Tento model říká, že složka může obsahovat soubory a vnořené složky. Lze přidávat podsložky a soubory do složky, odstraňovat podsložky a soubory ze složky, popřípadě získat odkaz na podsložku či soubor ve složce. Dále je možné zjistit velikost souboru či složky.
Aplikací návrhového vzoru COMPOSITE na tento model dostaneme výsledek znázorněný na obr. 4.



Obr. 4. – Model stromové struktury souborů a složek po aplikaci návrhového vzoru

Vzor COMPOSITE zavádí společné rozhraní (Prvek) pro uzly (tj. elementy stromu, které mohou obsahovat podřízené elementy neboli potomky, v našem příkladě třída Složka) a listy (tj. elementy, které prvky neboli potomky obsahovat nemohou, v našem příkladě třída Soubor). Použitím tohoto vzoru jsme navíc díky rozhraní Prvek sloučili obě agregační vazby v jednu (od třídy Prvek ke třídě Složka) a rovněž jsme zredukovali metody pro manipulaci s vnořenými složkami a soubory ve složce na metody pro manipulaci s vnořenými prvky.

Poznámka. Jelikož třída Soubor dědí metody pro práci s potomky z rozhraní Prvek a zároveň nemůže obsahovat potomky (jelikož se jedná o list stromu), je nutné implementovat tyto metody ve třídě Soubor tak, aby byl při jejich volání nějakým způsobem nastaven chybový stav (např. vyvoláním výjimky).

Opětovné použití návrhového vzoru COMPOSITE je možné například pro znázornění organizace tříd v jazyce Java do balíčků. V tomto případě budou uzly stromu představovat balíčky, zatímco listy budou reprezentovat třídy.
Aplikaci návrhového vzoru lze opět provést posloupností elementárních transformací:

1. Přidání rozhraní Prvek do modelu.
2. Přidání vazeb generalizace do modelu pro znázornění skutečnosti, že třídy Složka a Soubor implementují rozhraní Prvek.
3. Sloučení obou agregačních vazeb mezi směřujících od tříd Soubor a Složka do třídy Složka v jedinou agregační vazbu, která bude směřovat od rozhraní Prvek do třídy Složka. To obnáší rovněž následující:
   - Sloučení metod přidejPodsložku() a přidejSoubor() do metody přidejPotomka().
   - Sloučení metod odstraňPodsložku() a odstraňSoubor() do metody odstraňPotomka().
   - Sloučení metod vraťPodsložku() a vraťSoubor() do metody vraťPotomka().
4. Přidání metody přidejPotomka(), odstraňPotomka() a vraťPotomka() do třídy Soubor.
5. Přidání metod přidejPotomka(), odstraňPotomka(), vraťPotomka() a velikost() do rozhraní Prvek. Výsledný model odpovídá obr. 4.

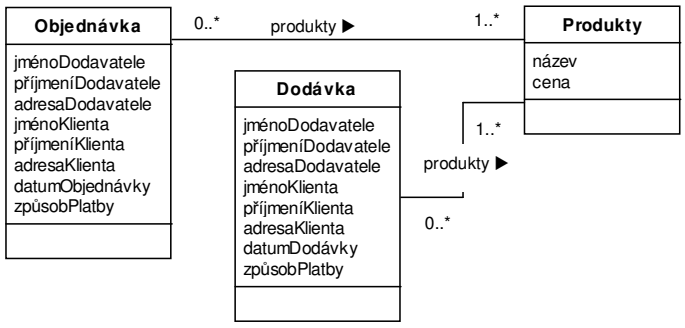
Z výše uvedených informací vyplývá, aplikace návrhového vzoru je transformace, která nemění chování systému. Lze tedy na ni nahlížet jako na refaktoring. Obecně však neplatí, že refaktoring je zároveň použití návrhového vzoru.

PŘÍKLAD TRANSFORMACE MODELŮ – OBJEKTOVÁ NORMALIZACE

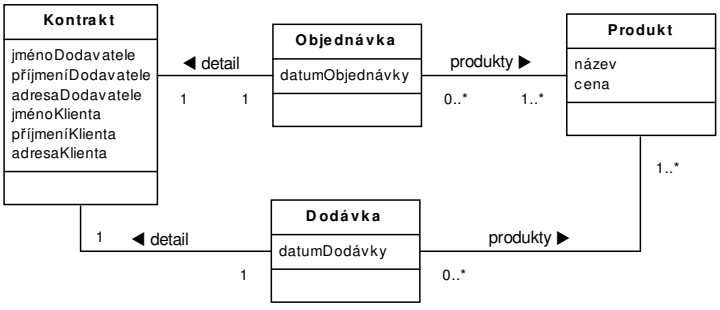
Objektová normalizace se týká objektově orientovaných datových modelů, tedy nikoliv objektového aplikačního modelu. Zavádí několik úrovní normálních forem a podmínky, která musí objektový datový model splňovat, aby byl v normální formě daného stupně.
Existuje řada přístupů k objektové normalizaci, z nichž některé lze najít v [3], kde je rovněž podrobněji rozebráno téma objektové datové modely. Pro účely tohoto příspěvku bude použit přístup autora [3], který definuje tři stupně normální formy objektového datového schématu následovně:

- Třída je v *první objektové normální formě* (1ONF), jestliže její objekty neobsahují skupinu opakujících se atributů. Schéma je v 1ONF, jestliže všechny třídy objektů v něm jsou v ONF.
- Třída je ve *druhé objektové normální formě* (2ONF), jestliže je v 1ONF a její objekty neobsahují atribut nebo skupinu atributů, které by byly sdílené nějakým jiným objektem. Schéma je v 2ONF, jestliže všechny třídy objektů v něm jsou v 2ONF.
- Třída je ve *třetí objektové normální formě* (3ONF), jestliže je v 2ONF a její objekty neobsahují atribut nebo skupinu atributů, které mají samostatný význam nezávislý na objektu, ve kterém jsou obsaženy. Schéma je v 3ONF, jestliže všechny třídy v něm jsou v 3ONF.

Příklad 3. Na obr. 5 a 6 jsou postupně znázorněny objektové datové modely v 1NF a 2NF. Tyto modely byly převzaty z [3], kde jsou rozebrány podrobněji.



Obr. 5. – Objektový datový model v první normální formě



Obr. 6. – Objektový datový model v druhé normální formě

Elementární transformační kroky přechodu od 1NF k 2NF jsou následující:

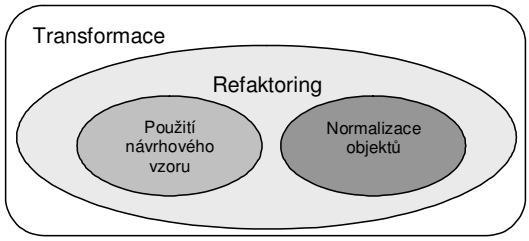
1. Přidání nové třídy Kontrakt do modelu.
2. Přidání asociačních vazeb 1:1 mezi třídy Kontrakt a Objednávka a třídy Kontrakt a Dodávka.
3. Přesunutí sdílených atributů týkajících se dodavatele a klienta ze tříd Dodávka a Objednávka do třídy Kontrakt. Výsledný model odpovídá obr. 6.

Transformaci modelu do normální formy o stupeň vyšší lze rozdělit na posloupnost elementárních transformací. Jedná se tedy o zvláštní případ refaktoringu.

SJEDNOCENÍ POHLEDU NA TRANSFORMACE

Z poznatků uvedených výše vyplývají následující tvrzení:
- Refaktoring je zvláštní případ transformace.
- Použití návrhového vzoru je zvláštní případ refaktoringu.
- Objektová normalizace je zvláštní případ refaktoringu.

Tato skutečnost je znázorněna graficky na obr. 7.



Obr. 7. – Vztah mezi refaktoringem, použitím návrhového vzoru a normalizací objektů

Každou transformaci lze rozložit na posloupnost kroků, z nichž každý představuje elementární transformaci. V souvislosti s tím je při každé transformaci potřeba zodpovědět následující otázky:
1. Zůstane po provedení elementární transformace chování systému nezměněno?
2. Je vybraná posloupnost elementárních transformací optimální z hlediska časové náročnosti?

FORMÁLNÍ POPIS TRANSFORMACÍ

Nejprve je potřeba definovat kalkul, který umožní formálně popsat jednotlivé transformace. Zavedeme pro něj název *transformační kalkul*. Je založený na kalkulu predikátové logiky a je doplněn o prvky, jejichž význam je uveden v tabulce 1.

Tabulka 1. – Rozšíření predikátového kalkulu o nové prvky transformačního kalkulu

Prvek kalkulu	Význam
$\mathcal{T}$ (*vstup*, *stav před*, *stav po*)	Zápis transformace. Transformaci chápeme jako uspořádanou trojici (*vstup*, *stav před*, *stav po*). *Vstup* je množina elementů, které do transformace vstupují. *Stav před* je množina podmínek, které jsou splněny před transformací, a stav po je množina podmínek, které jsou splněny po transformaci.
C	Množina všech tříd v daném modelu.
A	Množina všech atributů všech tříd v daném modelu.
M	Množina všech metod všech tříd v daném modelu.
A(*c*)	Množina všech atributů ve třídě *c*.
M(*c*)	Množina všech metod ve třídě *c*.
.name	Název třídy, atributu či metody.
$	Označení parametru transformace.
$\alpha(c_1, c_2)$	Třída c_1 je v asociaci se třídou c_2.

$\sigma(c)$	Předek třídy c.
$\iota(c)$	Množina všech rozhraní, která implementuje třída c.

V tabulce 2. jsou uvedeny příklady základních transformací a jejich formální zápis.

Tabulka 2. – Formální popis vybraných transformací

Transformace	Formální vyjádření
Přejmenování třídy	$\mathcal{T}(\{c.\text{name}, \$\text{newName}\}, \{c \in \mathbf{C}\}, \{c.\text{name}=\$\text{newName}\})$
Přesunutí metody do předka	$\mathcal{T}(\{c, m\}, \{c \in \mathbf{C}, m \in \mathbf{M}(c)\}, \{m \in \mathbf{M}(\sigma(c))\})$
Přesunutí metody do potomků	$\mathcal{T}(\{c_1, \text{m}\}, \{c_1 \in \mathbf{C}, m \in \mathbf{M}(c_1)\}, \{m \in \mathbf{M}(c_2)\ \forall c_2 \vert c_1 = \sigma(c_2)\})$
Vyčlenění metody do rozhraní	$\mathcal{T}(\{c, m, i\}, \{c \in \mathbf{C}, m \in \mathbf{M}(c)\}, \{m \in \mathbf{M}(i), i \in \iota(c)\})$
Vyčlenění atributů a metod do samostatné třídy	$\mathcal{T}(\{c_1, c_2, \text{E}\}, \{c_1 \in \mathbf{C}, c_2 \notin \mathbf{C}, \text{E} \subset \mathbf{A}(c_1) \cup \mathbf{M}(c_1)\}, \{c_2 \in \mathbf{C}, \text{E} = \mathbf{A}(c_2) \cup \mathbf{M}(c_2), (\text{E} \cap \mathbf{A}(c_1) \cup \mathbf{M}(c_1)) = \varnothing, \vert\alpha(c_1, c_2)\vert = 1\})$

ZÁVĚR

V předchozích odstavcích jsme nastínili souvislosti mezi transformacemi v objektově orientovaných modelech, refaktoringem, použitím návrhového vzoru a objektovou normalizací. Jádro těchto souvislostí spočívá v možnosti rozložit transformaci na posloupnost elementárních transformací, které lze formálně vyjádřit pomocí transformačního kalkulu. Formální popis může usnadnit implementaci transformací v CASE nástrojích či jiných aplikacích určených pro vývoj softwaru. Ne každá transformace je ovšem deterministická – v některých případech se uživatel bude muset rozhodnout, které prvky do transformace vstoupí. Otázkou zůstává, jak implementovat interpretaci transformačního kalkulu? Hledání odpovědi na tuto otázku může být předmětem dalšího výzkumu.

PODĚKOVÁNÍ

Tento příspěvek byl vytvořen za podpory grantu MŠMT 1P04LA211.

LITERATURA

1. Sunyé G. –Pollet D. – Le Traon Y. – Jézéquel J. M. *Refactoring UML Models.* http://www.irisa.fr/triskell/publis/2001/Sunye01b.pdf
2. Kraval I. – *Design Patterns v OOP se zaměřením na JAVU, C# a Delphi.* Elektronická publikace, 2002
3. Merunka V. *Normalizace v objektových databázích.* Sborník konference OBJEKTY 2004. Praha 2004.
4. Carda A. – Merunka V. – Polák J. *Umění systémového návrhu – Tvorba informačních systémů pomocí původní metody BORM.* Grada Praha, 2000, ISBN 80-247-0424-2
5. Garcia L. R: – Merunka V. – Polák J. BORM – *Business and Object Relation Modelling.* In: Proceedings of WOON – 6th international conference on object technology Institute of Electrotechnology, St. Petersburg, 2001, ISBN 80-88786-8
6. Merunka, V. *Modelování podle metody BORM pomocí nástroje Craft.CASE.* Sborník konference Objekty 2005.
7. Nouza, O. *Formální pohled na vztah mezi refaktoringem, návrhovými vzory a normalizací objektů.* Sborník konference Objekty 2006.