KRITIKA NĚKTERÝCH VÝKLADŮ OBJEKTOVĚ ORIENTOVANÉHO PARADIGMATU

Martin Molhanec

České vysoké učení technické – FEL, K-313, Technická 2, 166 27 PRAHA 6, Dejvice, Česká republika, tel.: (++420) 2 2435 2118
mailto: molhanec@fel.cvut.cz, http://martin.feld.cvut.cz/~molhanec

Abstrakt
Obsahem příspěvku je kritický pohled autora na výklad objektového paradigmatu, tak jak se s ním měl možnost setkat v pracích svých studentů, kolegů i v odborných publikacích. Speciálně bude kritice podroben chybný výklad objektového a entitě-vztahového paradigmatu, chápání modelu analytického a implementačního, atd.

1. Úvod

Tento příspěvek navazuje na moje předchozí příspěvky na této konferenci v minulých letech 1 a 2. Zdrojem všech mých příspěvků je moje nespokojenost s výkladem objektového paradigmatu a to zejména při výkladu objektově orientované metodiky, ale i při interpretaci klasických datově orientovaných principů. Nebezpečí vidím zejména u studentů, kteří nesprávným pochopením paradigmatu získávají nesprávné návyky a bude pak déle trvat nežli se jich zbaví. Netvrdím, že má tvrzení jsou vždy správná, ale i tak doufám, že možná vyvolají určitou reakci a diskuzi této problematiky.

2. Oblasti mé kritiky

Jaké jsou vlastně mnou kritizované oblasti? Pokusím se je nyní trochu neutříděnou formou naznačit.

- **Co je to vlastně analýza**
  Velké množství studentů stále ve své podstatě nechápe co je to vlastně analýza! Ano, ač většina z nich projde předměty *Softwarové inženýrství*, které také učím, stále se setkávám se skutečností, že mnozí studenti mají představu, že analýza je zbytečný výmysl (zejména s rozvojem tzv. *extrémního programování*, se tento trend stále zřetelněji objevuje), že jsou to jen nějaké obrázky, které dokumentují jejich programátorskou práci. Student prostě uvažuje takto, analýza jsou nějaké obrázky. Dobře – udělám tedy nejdříve program a podle toho programu udělám obrázky, které po mne chce můj pedagog a ty mu předložím jako výsledek! Tento přístup má pochopitelně dva zásadní nedostatky: za prvé je obráceno časové pořadí, nejprve se dělá program a pak „*analýza*“ a za druhé výsledné diagramy popisují implementaci nikoliv konceptuální pohled na problém!
- **Zakotvení v implementaci**
  Student je stále svým myšlením zakotven v implementaci. Přestože se tváří, že dělá analýzu a on si ovšem doopravdy myslí, že analýzu dělá, je myšlenkově stále pevně zakotven ve své implementační představě – objektově orientovaném programování v nějakém konkrétním jazyce, například C++, Smalltalku anebo ve využití nějaké

konkrétní relační či objektově orientované databáze. Proto je stále nucen si analytický model obohacovat konstrukty pro vyjádření svých implementačních nástrojů a nechápe, že v analytické oblasti jsou nepotřebné ba naopak nežádoucí!

- **Pochopení klasických datových modelů**
  Existují také chybné představy o entitě-vztahovém modelování (ERM: entity-relationship model). Důvod je ten, že se při vysvětlování ERM, ale i relačního databázového modelu (RDM: relational database model), pro lepší pochopení těchto datových paradigmat, využívají pojmy jako *tabulka* a *řádek v tabulce*, přestože se jedná o pojmy, které se v těchto paradigmatech nevyskytují! Proto je většina studentů a nejen studentů přesvědčena, že ERM je o nějakých tabulkách! Stejně dobře bych mohl tvrdit, že model objektově orientovaný (OOM: object oriented model), je o ukazatelích (*pointer*)!

3. Evoluce konceptuálních modelů dle Molhance ☺

Vzhledem k tomu, že mezi mými studenty, ale i některými kolegy, panuje dle mého názoru nesprávný vhled na jednotlivé modely, se kterými se běžně při konceptuální analýze pracuje, pokusím se v této kapitole tyto názory dle svého přesvědčení definovat a utřídit.

Především mluvme o konceptuálním modelování (CM: *conceptual model*) a nikoliv o datovém modelování (DM: *data model*). Důvod je ten, že CM je obrazem světa, který modeluje v jeho celé úplnosti. Konstrukty CM odpovídají skutečnostem ve světě kolem nás. DM je však obrácen k tomu, jak jsou uspořádána a strukturována data v některé datové implementaci (*databázi*), která se běžně využívá, například relační či objektově orientované databázi. Co z toho plyne?

- Konceptuální model je model světa okolo nás.
- Datový model je model uspořádání dat v databázi.
- V žádném konceptuálním modelu není konstrukt *tabulka*! Známé konceptuální modely využívají konstrukty: *entita* a *třída*.

Pokusím se výše uvedené teze vyjádřit uspořádáním všech zmíněných pojmů v následující tabulce.

zkratka	pojem česky	pojem anglicky	výklad	poznámka
CM	Konceptuální model	Conceptual Model	Konceptuální model je odrazem skutečného (reálného nebo abstraktního) světa kolem nás. V současné době se běžně využívají dva konceptuální modely: ERM a OOM.	Zkratka není běžně zavedena.
ERM	Entitě-vztahový model	Entity-Relationship Model	Entitě vztahový model podle *Chena* a dalších autorů. Původně vznikl na podporu datového modelování. Používá následující základní konstrukty: • Entita • Atribut • Vztah	
XERM	Rozšířený entitě-vztahový model	Extended Entity-Relationship Model	Rozšířený ERM rozeznává navíc dva typy vztahů: • Obecný • Specializace (dědičnost) Tento model například používá metodika IDEF1X a jiné.	Zkratka není běžně zavedena.

PERM	Fyzický ERM	Physical ERM	ERM omezený pouze na konstrukty, které se skutečně vyskytují v relační databázi	Zkratka není běžně zavedena.
OOM	Objektově orientovaný model	Object Oriented Model	Konceptuální model inspirovaný vznikem objektově orientovaného programování. Jeho základní konstrukty jsou: • Třída • Objekt • Atribut • Metoda • Vztah Rozpoznává několik typů vztahů: • Obecný • Skládání (kontejner, kompozice, agregace) • Dědičnost (specializace) Využívají ho například následující metodiky: OOA/OOD, OMT, UML, aj.	
DM	Datový model	Data Model	Model, který zachycuje vztahy mezi daty vzhledem k jejich konkrétní implementaci.	Zkratka není běžně zavedena.
RDM	Relační datový model	Relational Data Model	Datový model podle *Codda*. Jedná se o vysoce abstraktní model určený pro modelování relačních databází. Využívá základní konstrukty: • Relace • Atribut • Doména • Funkční závislost	
OODM	Objektově orientovaný datový model	Object Oriented Data Model.	Datový model určený pro modelování objektově orientovaných databází.	Zkratka není běžně zavedena.

Tab. 1

Rád bych upozornil na některé důležité skutečnosti.

- Je rozdíl mezi pojmy **Relation** a **Relationship**! První pojem je nutné překládat do češtiny jako *relace* zatímco druhý pojem jako *vztah* ! Tento rozdíl je naprosto **zásadní** pro pochopení rozdílu mezi ERM a RDM!
- Je také důležité pochopit, že pojem *entita* **nerovná se pojmu** *tabulka*! *Entita* je, podobně jako *třída*, množina *objektů* stejného typu! Pouze u PERM můžeme mluvit o tom, že *entita* ≈ *tabulka*.

Pokusím se popsat význam výše uvedených pojmů v následující přehledné tabulce.

pojem česky	pojem anglicky	výklad	poznámka
Vztah	Relationship *nebo také* Association	Vztah je nějaká souvislost mezi objekty konceptuálního modelu. Různé CM rozpoznávají různé typy vztahů.	Skutečnost, že několik anglických pojmů se překládá do českého jazyka jako jeden pojem přináší problémy při překladu!
Relace	Relation	Jedná se o konstrukt RDM. Podle obvyklé definice: *databázová relace je matická relace …*	Pozor – neplést s pojmem *relationship*!
Entita	Entity	Entita je základní konstrukt ERM. Podle obvyklé definice: *Entita je množina objektů stejného typu*.	Definice entity a třídy jsou ve své podstatě totožné. Rozdíl je ten, že ERM se zajímá pouze o data, OOM se zabývá i funkcionalitou.
Třída	Class	Třída je základní konstrukt OOM. Podle obvyklé definice: *Třída je množina objektů stejného typu*.	
Atribut	Attribute	Tento pojem se používá v RDM, ERM i OOM přibližně ve stejném smyslu, představuje datovou složku relace, entity nebo třídy.	
Metoda	Method	Pojem používaný v OOM, kde představuje funkční složku třídy.	
Objekt	Object	Jedna konkrétní instance třídy. Používá se v OOM.	U ERM mluvíme o tzv. *výskytu entity*, protože ERM nemá pojem podobný pojmu *objekt.*

Tab. 2

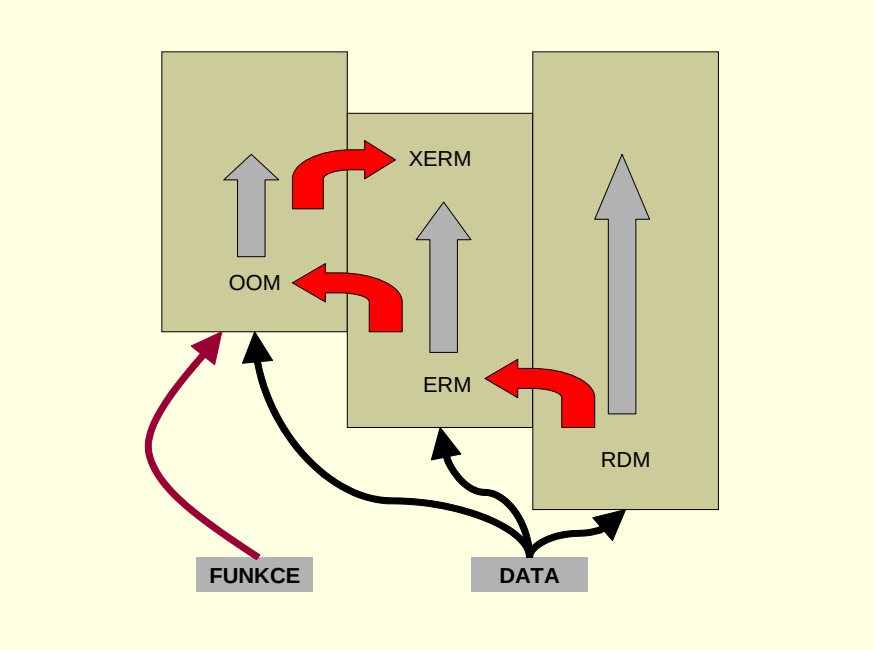
Opět považuji za vhodné upozornit na:

- *Entita* i *třída* jsou definovány jako množiny objektů stejného typu, zatímco *databázová relace* je definována jako *matematická relace*, nicméně i tak se vždy jedná o množiny.
- Všimněme si, že RDM odpovídá relační implementaci. Nikoliv však ERM!

Souvislosti mezi pojmy u různých modelů snad lépe objasní následující tabulka a Obr. 1.

ERM	XERM	PERM	OOM	RDM
Entita	Entita	Tabulka	Třída	Relace
Vztah (relationship)	Vztah (relationship)	Vztah (relationship)	Vztah (relationship)	Funkční závislost
	Obecný vztah (relationship)		Obecný vztah (association)	Funkční závislost
			Vztah skládání (kontejner, kompozice, agregace) (container, composition, aggregation)	
	Vztah dědičnosti (inheritance, specialization)		Vztah dědičnosti (inheritance, specialization)	
Výskyt entity	Výskyt entity	Řádek tabulky (raw)	Objekt (object)	n-tice
			Metoda (method)	

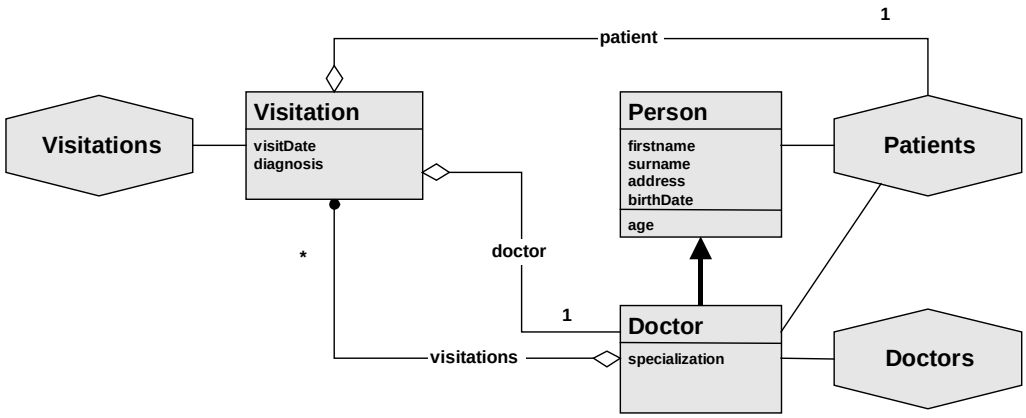
Tab. 3



Obr. 1

4. Množiny, které se příliš množí

Tato část mého příspěvku je reakcí na koncepci mého váženého kolegy Vojty Merunky 3, který si začal do svých diagramů kreslit konstrukt *množiny* a zastává názor, že se jedná o významný krok vpřed. A nejenže si konstrukt množiny kreslí on sám, ale ještě to učí své studenty! Podívejme se nejprve na jeden jeho diagram (Obr. 2). Jedná se *implementační diagram objektově datového modelu* (dle notace kolegy Merunky), který odpovídá mému OODM v Tab. 1.



Obr. 2

Jak je zřejmé, množiny jsou znázorněny pomocí šestiúhelníků. Připomeňme si však následující skutečnosti.

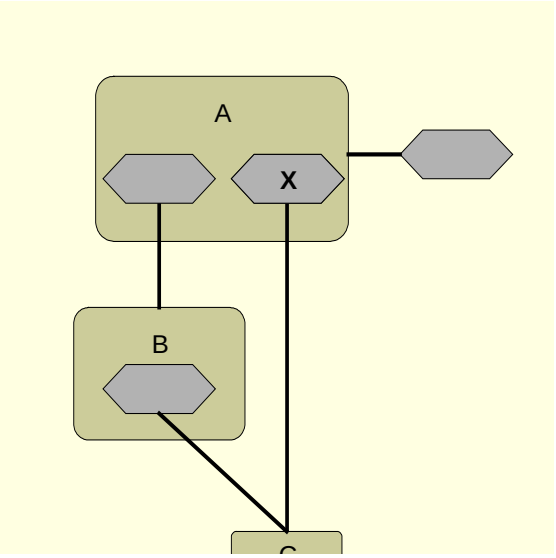
- Klasický výklad konstruktu *entita* a *třída* v diagramech, které jsou používány v ERM a OOM je ten, že daný konstrukt představuje, jak definici daného konstruktu, tak i množinu jeho instancí. Koneckonců, toto je naprosto přirozený výklad. Je zřejmé, že samotná definice je nepotřebná, pokud neuvažuji o jejím zpředmětnění

prostřednictvím jejích instancí. A pokud si vytvářím nějaké instance, tak jenom proto, abych s nimi pracoval v rámci nějaké množiny.

- V souvislostech výše uvedeného výkladu je dle mého názoru zřejmé, že pokud každé *entitě* nebo *třídě* náleží právě jedna množina, není nutné si tuto defaultní množinu nikterak zakreslovat. Ostatně i v PERM bychom mohli uvažovat tak, že ke každé *entitě* přísluší jedna *tabulka* a diagram si tak kreslit!
- Domnívám se, že jeden důvod pro kreslení *množin* je dán tím, že při objektovém databázovém přístupu je v některých nástrojích (například *Smalltalk DB*, který používá kolega Merunka) oddělena *definice* a *realizace* datového úložiště. Nicméně v obvyklých relačních databázích je *definice* a *realizace* datového úložiště obsažena v jediném SQL příkazu *CREATE TABLE*, proto zřejmě datově orientovaní modeláři nikdy neuvažovali, že by si měli ke každé konceptuální entitě kreslit její relační implementaci - tabulku.
- Může být ovšem namítnuto, že výše uvedené zdůvodnění neplatí pro vztah mezi třídou *Doctor* a množinou *Patients*. Pokud se ovšem zamyslíme nad způsobem, jak rozumně pracovat s dědičností, je nasnadě, že pokud je třída *Doctor* specializací třídy *Person*, ukládání potomků do množiny, která mu přináleží a současně do množiny, která přináleží jeho předku je opět defaultní.
- Na základě výše uvedené úvahy je možné odstranit i tento vztah a výsledný diagram nebude obsahovat žádné zbytečné množiny *a to je to, co jsem chtěl dokázat!*
- Podobné úvahy je možné uskutečnit i s ohledem na modelování dalších typů vztahů (obecný vztah a skládání).

Nechci, aby tato kapitola vyzněla jako *absolutní* kritika zavádění konstruktu *množina* do *objektově orientovaného datové modelu*. Pokusím se vyslovit následující teze.

- V úrovni konceptuálního modelování není konstrukt *množina* zapotřebí, protože se jedná o *implementační* konstrukt.
- V úrovni modelování struktury dat v objektově orientované databázi je nepotřebný pokud předpokládáme *defaultní* implementaci. Uznávám ovšem, že můj výše uvedený důkaz není zcela jistě úplný a bude tedy jistě podroben vášnivé kritice.
- Vzhledem k tomu, že některé konceptuální vztahy je možné v OODBMS implementovat více možnými způsoby 6, a to způsobem *relačním* nebo *objektovým,* a nebo je možné realizovat i množiny, které jsou nad rámec defaultních množin, je způsob kreslení diagramů, který navrhl kolega Merunka, jistě užitečný! Nicméně si nejsem jist, zdali je tento návrh kompatibilní se stávajícím standardem UML 4 a 5. Bohužel, bez využití současných mezinárodně přijatých standardů nelze očekávat úspěšné všeobecné přijetí navrhované notace.



Obr. 3

5. Normalizace je stará dobrá známá

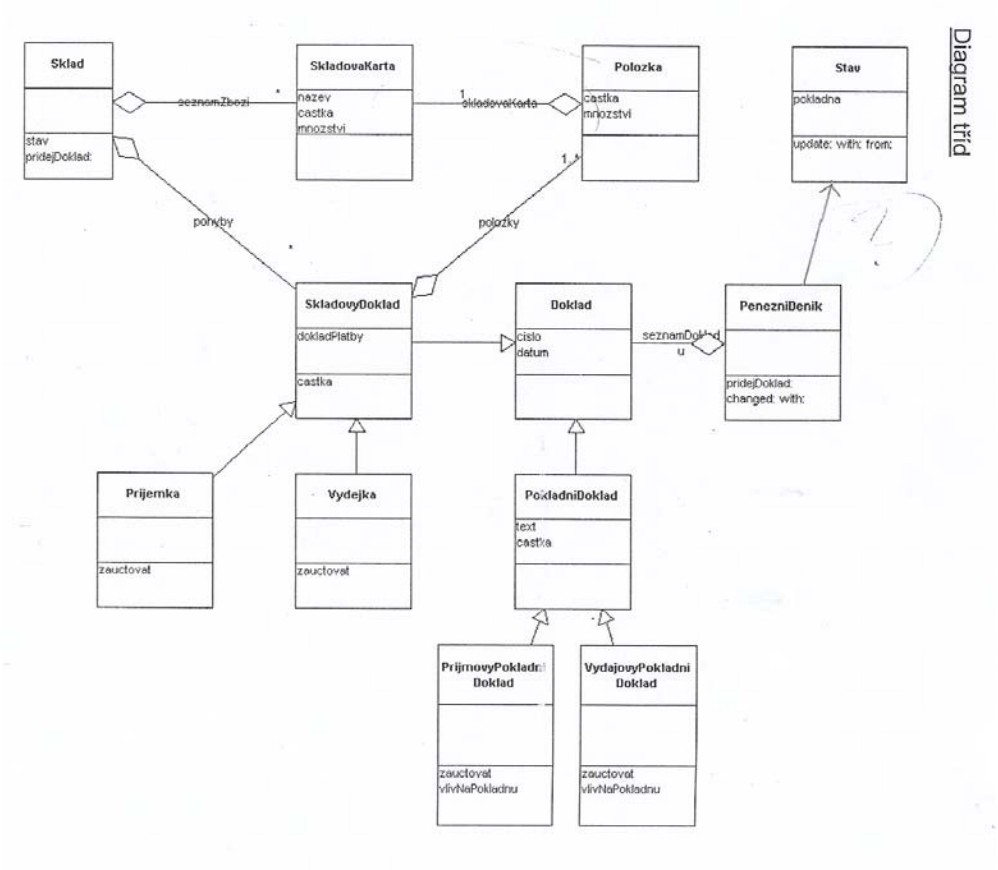
V této krátké kapitole chci upozornit na jednu skutečnost, která na první pohled není zřejmá, ale měla by na ní být upřena pozornost. V oblasti teorie databázových systému (RDM) je *normalizace* obsáhlá a poměrně náročná látka, která vychází z koncepce *funkčních vztahů*. Jejím cílem je navrhnout *správné* relace bez *nadbytečných* (*redundantních*) atributů. Hovoříme pak o 1. až 5. normální

formě a několika dalších. Za správně navrženou databázi se pak běžně považuje databáze v 3. normální formě, protože dosažení vyšších normálních norem je již prakticky obtížně dosažitelné. Ale je tomu tak i u objektových databází? Existují i pro ně normální formy ? A jsou vůbec potřebné? Já i můj kolega Merunka se domníváme, že ano!
Ukažme si základní problém normalizace na jednoduchém příkladě, který je na Obr. 3. Vidíme, že existují tři třídy objektů: A, B a C. Vztah mezi třídami A a B je vztah kontejneru a mezi třídami B a C taktéž. Každá třída má k sobě odpovídající množinu do které ukládá své objekty. Vzhledem k tomu, že se jedná o vztah kontejneru (skládání) jsou množiny pro třídy B a C zakresleny uvnitř tříd A a B. To je vše v pořádku. Nicméně programátor se rozhodl, že pro rychlejší přístup k objektům třídy C bude je ukládat taktéž do množiny **X**, která je uvnitř třídy A! Je jasné, že se jedná o nadbytečnost, ale na straně druhé tato nadbytečnost může funkci programu s touto strukturou dat významně urychlit při některých operacích! U klasické relační implementace bychom věděli, že podobná struktura odporuje některé normální formě a že by to tak být nemělo! Ale co o tom praví teorie objektových databází?

Naštěstí existují pokusy, nicméně nepříliš známé a všeobecně zatím neuznávané o vyřešení problému normalizace v OODBMS, například 7.

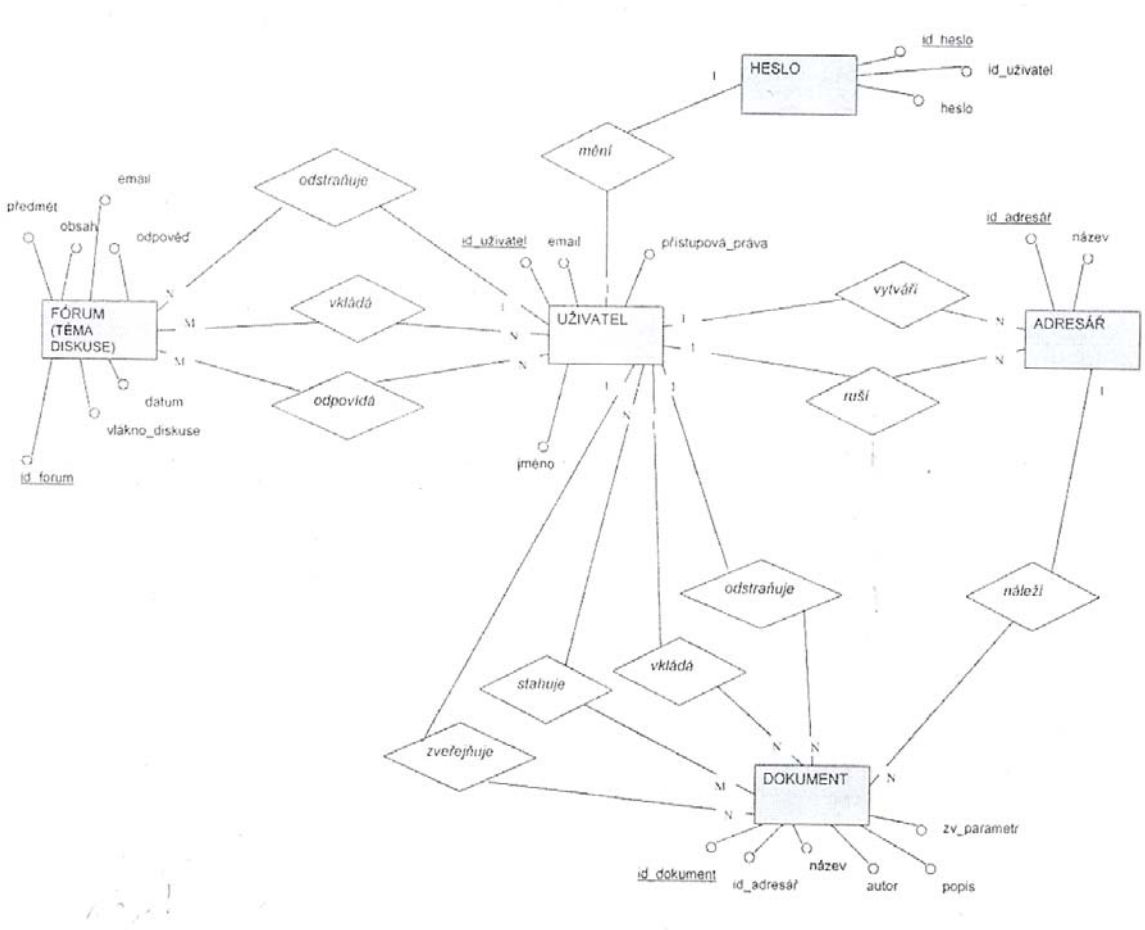
6. Příklady studentských chyb

Na závěr svého článku uvedu několik diagramů z diplomových prací, ve kterých shledávám chyby. U každého příkladu velice stručně vyjmenuji, které skutečnosti považuji za chybné.

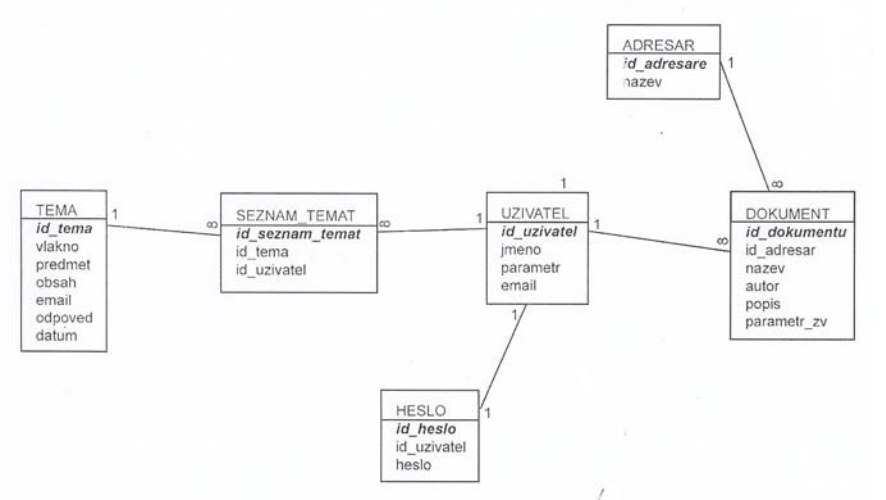


Obr. 4

- Vztah se šipkou mezi *PenezniDenik* a *Stav* je podivný. V konceptuálním diagramu se přeci žádné šipky nepoužívají. Je to vliv četby špatných knížek o UML 2.
- Vztah mezi *SkladovaKarta* a *Polozka* je obráceně. Zřejmě nepozornost studenta.
- Nadbytečný vztah mezi *Sklad* a *SkladovýDoklad*. Typický problém normalizace, tak jak byla diskutována v kap. 5.

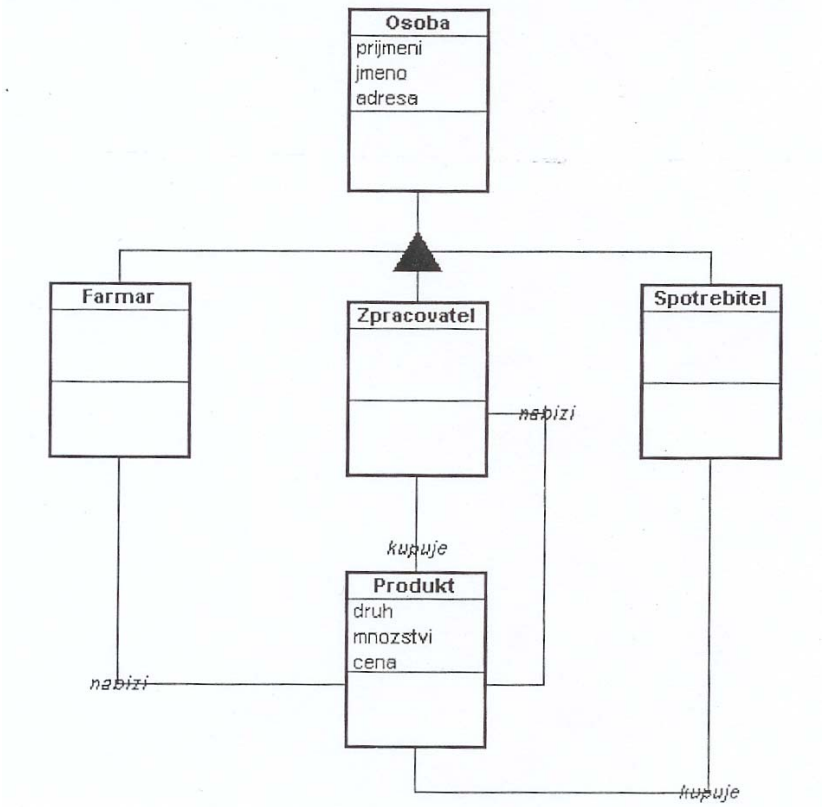


Obr. 5

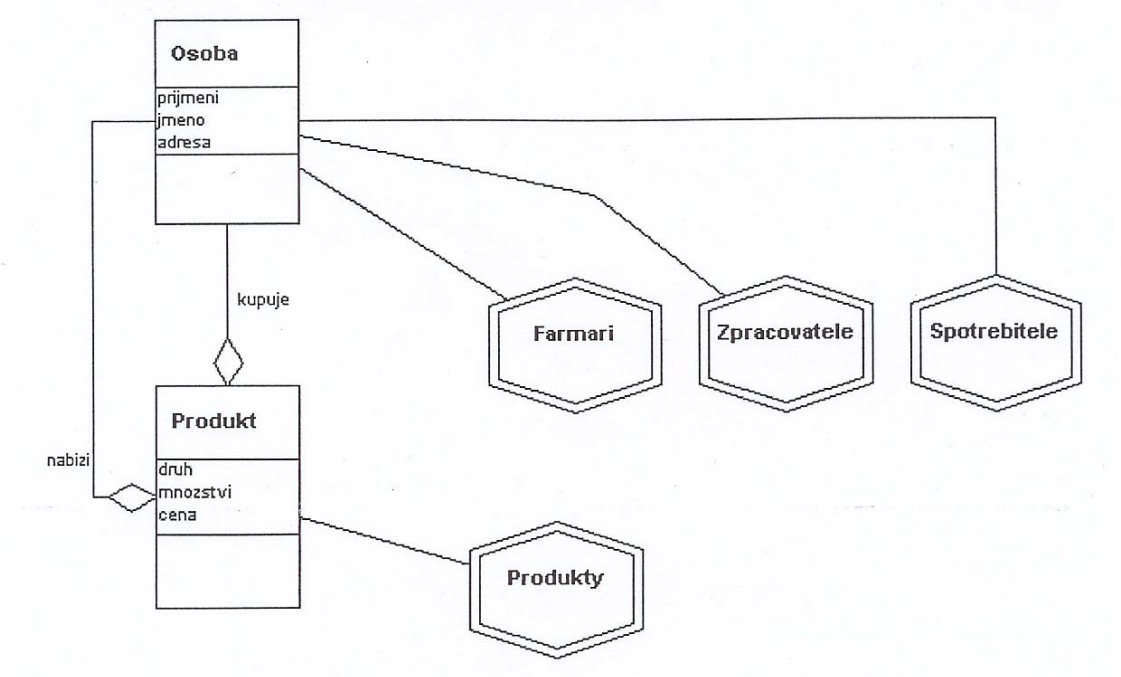


Obr. 6

- Na prvním diagramu se vyskytuje nesmyslné množství vztahu (například: zveřejňuje, stahuje, ukládá, odstraňuje), které nemají žádné opodstatnění. Nejedná se o vztahy, ale o stavy, funkce nebo práva.
- Z druhého diagramu je zřejmé, že vztahy na prvním diagramu byly nesmyslné, protože se na něm nevyskytují. Druhý diagram není evolucí, upřesněním ani rozumnou transformaci prvého.



Obr. 7



Obr. 8

- První diagram je naprosto nedostatečný. Diagram, kde není zakreslena kardinalita a parcialita, vypovídá o polovině skutečností!
- Ve druhém diagramu se nám najednou vztahy upřesňují. Dle autora se jedná o vztah kontejneru, ale je nakreslen obráceně!

7. Shrnutí

Přestože je objektově orientované paradigma známo již delší dobu (od počátku 80 let a jeho rozkvět nastal v 90 letech), nejsou stále vyřešeny všechny jeho aspekty. Vznik a používání OODBMS vede k úvahám o normalizaci v objektově orientované oblasti. Jak je z mého článku patrné, neexistují ani stejné názory na souvislosti mezi jednotlivými konceptuálními modely a mnozí autoři je používají velmi rozdílně. Můj článek se zabývá pouze některými oblastmi OO paradigmatu, které mne v poslední době zaujaly. Celé obrovské oblasti, například souvislost mezi funkčním a konceptuálním modelem, nebyly v tomto článku vůbec zmíněny natož diskutovány.

Podobně byly naznačeny pouze některé chyby, které při práci s OOM vytvářejí studenti. Velice slabě byly naznačeny příčiny těchto chyb. Je ovšem těžké vytýkat chyby, když ani mezi autory nejsou na některé aspekty stejné názory. Nicméně existují určité základní skutečnosti, které by měly být vykládány vždy stejně a studenti by jim měli správně rozumět.

Doufám, že můj článek bude příspěvkem do odborné debaty o všech aspektech objektově orientovaného paradigmatu.

Literatura:

1. Molhanec, Martin: „Objektové metodologie – jejich užití a výklad“, Tvorba software 2003, TANGER, Ostrava 2003
   On line: http://martin.feld.cvut.cz/~mmm/VaV/files/publik/2003/OOkrit-co.pdf
2. Molhanec, Martin: „UML – několik kritických poznámek“, Tvorba software 2002, TANGER, Ostrava 2002
   On line: http://martin.feld.cvut.cz/~mmm/VaV/files/publik/2002/UML.pdf
3. Merunka, Vojtěch: „Objektový přístup v databázové technologii“, Tvorba software 2004, TANGER, Ostrava 2004
4. Booch, G., Jacobson, I., Rumbaugh, J.: „The Unified Modeling Language Reference Manual“. ADDISON-WESLEY, 1999. ISBN 0-201-30998-X
5. Booch, G., Jacobson, I., Rumbaugh, J.: „The Unified Modeling Language User Guide“. ADDISON-WESLEY, 1999. ISBN 0-201-57168-4
6. Kroha P.: „Objects and Databases“. McGraw Hill, London 1995, ISBN 0-07-707790-3
7. Dobbie Gillian: „Towards normalization in object-oriented databases“, Technical Report CS-TR-96/16. Victoria University of Wellington, Department of Computer Science. On line: http://www.mcs.vuw.ac.nz/comp/Publications/archive/CS-TR-96/CS-TR-96-16.pdf