Postup při objektově orientovaném návrhu ve Smalltalku

František Huňka

Ve svém příspěvku bych chtěl prakticky ukázat postup při práci s objektově orientovaným systémem. Celou problematiku chci uvést na jednoduchém systému řízení knihovny. Jako objektově orientovaný systém jsem si vybral Smalltalk/V.

Výhodou Smalltalku je většinou přímé převedení abstraktních popisů do výrazů ve Smalltalku, nebo jejich transformace jen s malou modifikací. Implementace se provádí změnou existujících metod ve třídách, nebo deklarací nových tříd. Takto je vlastně již začleněno velké množství požadovaného chování do existující dědičné hierarchie bez nutnosti dalších kroků.

Náš postup při vývoji tohoto problému byl následující:

- vytvoření *konceptuálního modelu*, který je složen z objektů, jejich základních vazeb a činností. *Konceptuální model* jsme odvodili z funkčních požadavků na uvedený systém.

- převedení *konceptuálního modelu* do externí specifikace ve Smalltalku. To provedeme pomocí vytvoření tříd, které korespondují typům objektů v *konceptuálním modelu* a které mají zprávy, pomocí nichž *konceptuální model* provádí základní operace.

- implementace tříd a objektů popsaných externími specifikacemi, včetně metod, které implementují zprávy vyžadované externími specifikacemi.

Vytvoření *konceptuálního modelu* není v terminech objektově orientovaného přístupu lehký problém, zvláště když se nabízí řada postupů. Naším cílem však není řešit tento problém, ale podívat se na *generické metody*, což je podle našeho mínění dobrý startovní bod. Generičnost (*genericity*) je schopnost definovat parametrické moduly. Příkladem je např. typ seznam, kde seznam může být seznam jmen, nebo seznam celých čísel. Vlastností generického kódu je nezávislost na typu a datech.

Když převádíme *konceptuální model* externí specifikace, měli bychom se zaměřit na několik základních abstraktních datových typů a upřesnit, jak je co možná nejvíce upotřebíme jako instance, proměnné a metody. Základní funkce těchto tříd budou zděděny ze základních typů a zbytek funkčních požadavků bude realizován přidáním dalších tříd a potřebných metod k rodičovským třídám. Tento problém je známý pro programátory ve Smalltalku. Spíše se jeví problémem, jak si z toho velkého množství tříd a metod vybrat nejvhodněji.

Funkční požadavky na systém

Systém řízení knihovny by měl zabezpečovat tyto funkce:

- přidání/zrušení knihy do/z knihovny, editace informací o knihách
- přidání/zrušení čtenáře do/z katalogu čtenářů, editace informací o čtenářích
- udržování informací o všech knihách v knihovně, povolení uživatelům prohledávání těchto informací a vyhledání informací podle titulu knihy a autora

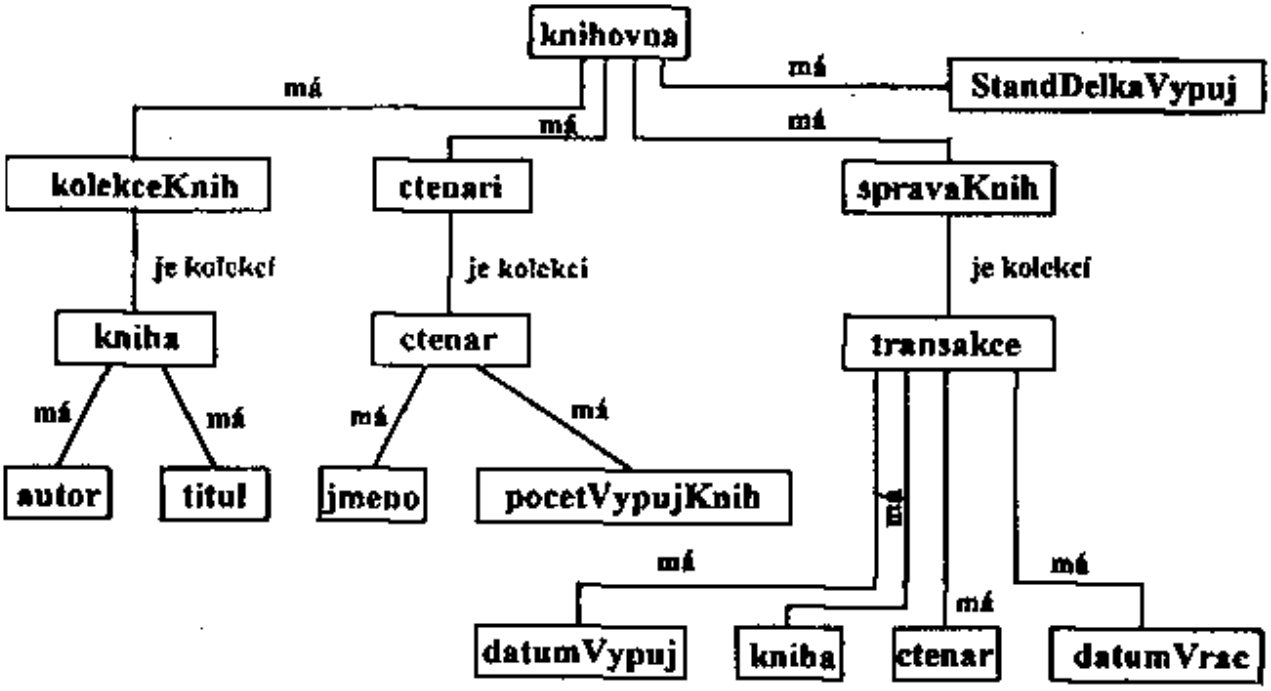
- provádění kontroly jednotlivých výpůjček a kontroly návrat knih do knihovny
- vytváření zprávy o stavu knihovny, což obsahuje celkový stav knihovny, seznam vypůjčeních knih a knih nevrácených v termínu.

Základní konceptuální model

Nyní potřebujeme navrhnout základní objekty, které odpovídají požadavkům na tento systém a definovat vztahy mezi těmito objekty.

Základní objekty by měly zahrnovat:

- knihovna
- čtenáři knihovny (všichni zájemci o služby knihovny)
- kolekce knih (souhrn všech knih knihovny)
- kniha
- čtenář



Obr. 1 Základní model

Navíc ještě potřebujeme objekt, ve kterém budou uloženy záznamy o transakcích výpůjček. Z analogie můžeme tento objekt nazvat *správaKnih* a objekty *transakce*. Ačkoli ještě samozřejmě budeme muset přidat pár drobností, náš základní *konceptuální model* je již hotov. Jeho schéma je na obrázku č. 1. Další objekt, který se na obrázku je *StandDelkaVypuj*, což představuje maximální délku výpůjčky. Aby tato informace byla pro všechny třídy přístupná a snadno modifikovatelná, proto je umístěna hned pod *knihovnou*.

Seznam objektů a jejich vzájemných vztahů je dobrý začátek, ale ještě stále nevíme, co budou objekty dělat, tedy jaké akce budou provádět a jaké metody k tomu budou potřebovat. K tomuto účelu je výhodné specifikovat základní činnosti pro knihovnu jako objekt. Tyto základní činnosti musí pokrývat základní požadavky na knihovnu.

Základní činnosti jsou:

- půjčit knihu čtenáři
- převzít knihu od čtenáře
- přidat/zrušit čtenáře do/z registru čtenářů
- přidat/zrušit knihu do/z katalogu knihovny
- vypsat zprávu o současně vypůjčených knihách a o knihách vypůjčených přesčas
- vyhledání knihy podle autora nebo titulu

Externí specifikace tříd pro knihovnu

Při této specifikaci se budeme snažit hledat pro každý navržený objekt nejbližší třídu ve Smalltalku, která bude nejblíže plnit jeho požadavky.

Začneme s třídou nejvyšší, tedy knihovnou. Jak je definováno v *konceptuálním modelu*, *knihovna* obsahuje nebo vlastní *kolekceKnih*, *čtenáři* a *správaKnih*. Ale sama o sobě není žádným z těchto objektů. Zdá se, že každý z těchto vztažených objektů bude ve vztahu ke *knihovně*, jako instanční proměnná ke své třídě. Ale nejobtížnější části objektově orientovaného návrhu je, do které třídy zařadíme hlavní objekt. Častokrát je třída vybrána jako existující třída tak, aby co nejvíce vyhovovala požadavkům abstraktních operací. Většinou zde bývají buď soupeřící třídy, nebo žádný kandidát.

Objekt knihovna vykonává většinou administrativní funkce. Vydává zprávy o kolekci knih, společnosti čtenářů, a dále pak provádí skutečné práce s přidáním/rušením knihy, půjčením/návratem a prohledáváním kolekcí knih a čtenářů. Objekt knihovna nemá žádné specifické funkce týkající se speciálních operací, nebo grafických operací. Protože je zde absence jakýchkoli důvodů někam přiřadit objekt *knihovna*, zařadíme ji přímo pod třídu *object*. Viz obr. 2.

Zatím nejsou definovány žádné instanční ani *class* metody. Všimněte si, že instanční proměnná *maxDeklaVypuj*, která uchovává maximální půjčovací periodu (např. 1 měsíc), jsme také přiřadili *classVariable StandDenniVypujcka*, která slouží jako základní standardní hodnota pro instanční proměnné. Je to proto, že *classVariable* jsou sdílené všemi instancemi dané třídy.

Nyní přistoupíme k vytváření metod pro instance této třídy. Tyto metody budou patřit do několika kategorií:

- práce s knihami — *přidání/rušení knih z kolekce*
- práce se čtenáři — *přidání/rušení čtenáře ze katalogu*
- prohledávání — *vyhledání konkrétní knihy nebo čtenáře*
- vypracování přehledu — *vytvoření seznamu vypůjčených knih a nevrácených knih*

Máme zde navíc metodu, která nebyla deklarována v *konceptuálním modelu*. Je to prohledání seznamu čtenářů podle zadaného jména.

Doposud jsme definovali protokoly pro instance třídy *Knihovna*. Pro vytvoření instancí musíme doplnit *class method* o metodu new, která vrátí nový objekt knihovny s prázdnou kolekcí *kolekceKnih*, *ctenari* a *spravaKnih*.

Pokud se podíváme na tři instanční proměnné třídy knihovna (*kolekceKnih*, *ctenari* a *spravaKnih*), musíme říci, že všechny tyto instanční proměnné musí být kolekcemi objektů. Smalltalk sám obsahuje s použitím *Class Browseru* řadu hierarchicky uspořádaných "kolekcí". První patrně

zřejmá možnost by byla vybrat třídu *Array*. Pokud bychom si tuto třídu vybrali, naše instance by zdědila všechny vlastnosti (metody) uvedené třídy. Například metoda pro přidávání prvku, by musela nalézt vhodné místo v poli, otestovat, zda toto místo je volné a pokud by to bylo třeba, udělat si místo (posunem všech prvků pole o jedno místo dolů). Podobně i rušení prvku by přineslo podobné potíže, což by vyžadovalo posun zbývajících prvků pole zase o jedno místo.

Na druhou stranu třída *OrderedCollection*, která je potomkem třídy *Array*, by mohla splňovat více naše požadavky. Tato třída při podrobnějším prohledání má mnoho společných funkčních charakteristik jako třída *Array* (což je ostatně logické, protože je jejím potomkem). Avšak instance třídy *OderedCollection* řídí své prvky jinak, než třída *Array*. Obě třídy zpřístupňují své prvky použitím celých čísel jako externích klíčů. Ale instance třídy *OrderedCollection* to provádí tak jak uživatel přidává, nebo ruší prvky kolekce. To znamená, že tato třída má nezbytné vlastnosti k implementaci zásobníku, nebo fronty.

(Metody *add: addFirst: addLast: removeFirst removeLast* umožňují snadno pracovat s instancí třídy *OrderedCollection* jako se zásobníkem nebo s frontou).

Pokud by kniha, nebo čtenář měli jedinečný identifikátor (klíč), pak by bylo výhodné použít k reprezentaci třídu *Dictionary* (slovník). To by nám umožnilo velmi rychlý přístup k informacím. Avšak protože knihy nemají jedinečný identifikátor stejně jako čtenáři, navíc prohledávání provádíme nejen pouze podle názvu knihy resp. čtenáře, ale také podle dalších možných parametrů, zůstaneme u třídy *OrderedCollection*, která naše požadavky bude plnit nejlépe.

Další možná třída *SortedCollection* by na první pohled mohla plnit naše požadavky ještě lépe, ale pro nás bude výhodné, nechat si pouze vybranou kolekci setřídit, nemít ji setříděnou neustále.

Tvorba kostry aplikace

Obr. 2 Struktura tříd.

```
Object subclass: #Knihovna
instanceVariableNames: 'kolekceKnih ctenari
                spravaKnih maxDelkaVypuj'
classVariableNames: 'StandDelkaVypuj'
poolDictionaries: ''

OrderedCollection subclass: #KolekceKnih
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''

OrderedCollection subclass: #Ctenari
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''

Object subclass: #SpravaKnih
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
```

```
Object subclass: #Transakce
instanceVariableNames: 'knika ctenar
        datumVypuj datumVrac'
classVariableNames: ''
poolDictionaries: ''

Object subclass: #Ctenar
instanceVariableNames: 'jmeno pocetVypujKnih'
classVariableNames: ''
poolDictionaries: ''

Object subclass: #Kniha
instanceVariableNames: 'autor titul umisteni'
classVariableNames: ''
poolDictionaries: ''
```

Vytvoření nové knihovny s prázdnými kolekcemi instančních proměnných pak provedeme pomocí *class method* new:

```
|aKnihovna|
aKnihovna := super new.
aKnihovna kolekceKnih: KolekceKnih new.
aKnihovna ctenari: Ctenari new.
aKnihovna spravaKnih: SpravaKnih new.
aKnihovna maxdelkaVypuj: StandDeklaVypuj.
^aKnihovna
```

Při tvorbě aplikace ve Smalltalku musíme pamatovat na jednu věc a to, že názvy metod bývají totožné s názvy proměnných. Tak např. řádek
aKnihovna kolekceKnih: KolekceKnih new.
kolekceKnih: je metoda,
KolekceKnih je název třídy, který slouží jako argument v předešlé metodě.

Interaktivní aplikace

Samostatným problémem v objektově orientovaných systémech jsou vstupně/výstupní operace. K jejich realizaci se používá MVC model (model-view-controller). Aby se mohli zachovat hlavní přednosti objektově orientovaného přístupu (adaptibilita a znova použitelnost) celý systém musíme rozdělit do tří tří funkčních celků:

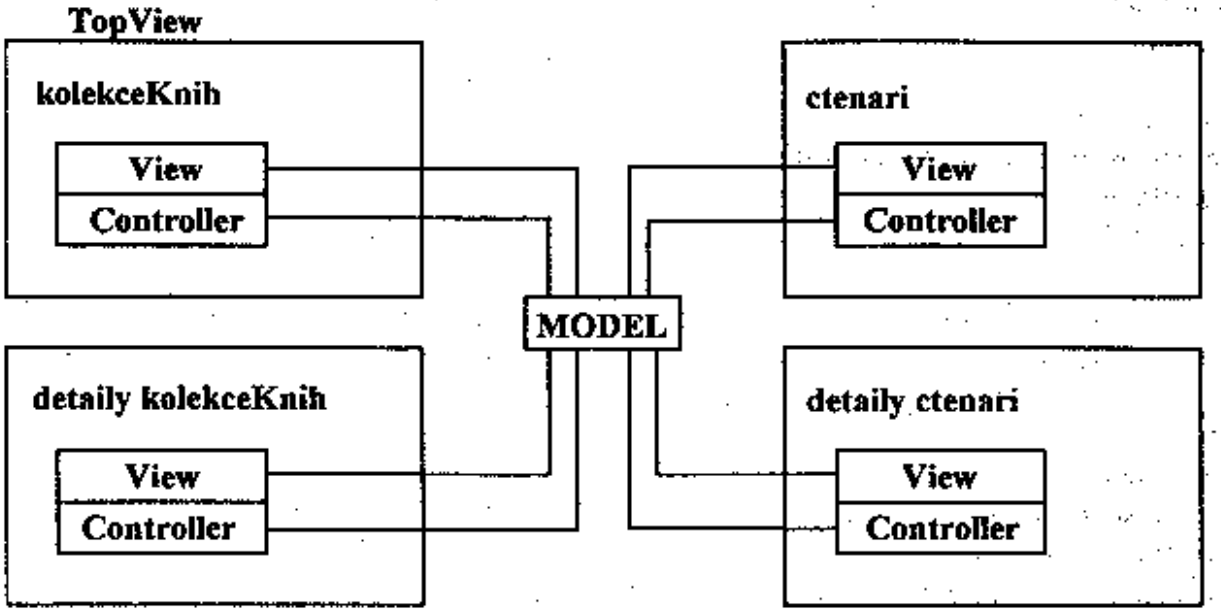
- model reprezentuje vlastní řešení systému
- view představuje všechny výstupy (pohledy) na model, během celé aplikace
- controller zpracovává uživatelské vstupy (klávesnice a myš) zasláním odpovídajících zpráv modelu nebo i view (pohledu). Controller představuje spojovací článek mezi modelem a view.

Nezbytným rysem systému MVC je jeho schopnost zobrazovat všechny pohledy průběžně aktualizované, odpovídající změnám modelu. Toto je zabezpečeno implementací metod, které patří pod třídu *object*. Mezi *modelem* a *view* existují závislosti.

Každý objekt zdědí od třídy *Object* schopnost určit závislé objekty. Když je nějaký objekt specifikovaný jako závislý, je přidán do kolekce závislých objektů. Kdykoli si objekt přeje oznámit závislým objektům změnu, pošle si sám sobě zprávu *changed*.

Konkrétní tvorba pohledů pak závisí na skutečných požadavcích utvářeného systému.

Obr. 3 Struktura pohledů



Při trochu složitější aplikaci samozřejmě nevystačíme s jedním pohledem, ale musíme vytvořit složený pohled. V našem případě se celkový pohled skládá z pohledu na kolekci knih a pohledu na čtenáře.

Závěr

Výhodou při práci s plně objektově orientovaným systémem je, že poskytuje hierarchii tříd, která nám urychlí vlastní aplikaci. Je možno zvolit mezi dědičným přístupem a přístupem *klient-server*. Trochu nezvyklý je model MVC, (organizace vstupně výstupních operací). Pokud však máme již nějaké zkušenosti s *event-driven-programming*, pak i to pro nás nemusí být složitý problém.

Literatura:

[1] Goldberg,A.: Smalltalk-80: The Language and Its Impementation, Reading, MA: Addison-Westley, 1983
[2] Smalltalk/V Manual: Tutorial and Programing Handbook
[3] Gray,F.D.: Smalltalk-80: A Practical Introduction Pitman Publishing, 1990

Autor: Ing.František Huňka,CSc.
katedra informatiky a počítačů
Přf, OU Ostrava
Bráfova 7, Ostrava 1
e_mail: hunka@oudec.osu.cz