VZORY PRO NÁVRH

V.Snášel, V.Sklenář

1. Úvod

Zkušený návrhář navrhuje řešení s dobrým designem. Je třeba poměrně dlouhé doby k tomu, aby začátečník získal znalosti nutné k dobrému návrhu softwarového produktu. Zkušený návrhář evidentně zná něco co nezná začátečník. Položme si otázku co je to?

Zkušený návrhář neřeší svůj návrh "od nuly". Raději znovu používá řešení, která již fungují. Jestliže najde dobré řešení používá ho ve své práci stále znovu a znovu. Tyto zkušenosti z něj dělají experta. Postupně nachází opakující se vzory tříd a způsob komunikace mezi objekty, je schopen je zobecnit a implementovat v různých objektově orientovaných systémech. Tím se jeho řešení stávají flexibilní, elegantní a ve vysoké míře znovupoužitelné. Opakované používání úspěšných řešení značně zvyšuje produktivitu práce při návrhu nových systémů. Vliv na jejich kvalitu je taktéž nezanedbatelný.

Všichni známe hodnotu návrhářské zkušenosti. V mnoha případech při řešení designu máme déja-vu - pocit jako kdybychom problém řešili již dříve, ale nevíme přesně kdy a jak. Proto by bylo dobré zaznamenat zkušenosti s návrhem pro další použití.

Cílem tohoto článku je popsat způsob záznamu zkušeností získaných při OOD pomocí vzorů pro návrh (Design Patterns). Každý vzor pro návrh je dobré pojmenovat, popsat (vysvětlit) a zhodnotit. Takto zaznamenané vzory pro návrh umožňují snadné znovupoužití úspěšných návrhů a architektur. Dalším důležitým důsledkem použití vzorů pro návrh je zlepšení dokumentace celého systému a předávání know-how mezi návrháři. Ve vzorech jsou vlastně uchovány užitné vlastnosti programových řešení. Vzory tyto vlastnosti pojmenovávají a tím umožňují vytvářet katalog vzorů. Při návrhu nových systémů se pak využívá katalog takto zaznamenaných problémů a jejich řešení.

Rozvoj této metodologie začal počátkem devadesátých let a mezi její hlavní propagátory patří řada významných osobností spojených s OOP, jako např. G.Booch a J.O.Coplien. Na konferenci OOPSLA '94 pořádané ACM byla této metodologii věnována velká pozornost a v letošním roce patří programování s využitím vzorů (pattern programming) mezi hlavní body konference.

2. Co znamená pojem "Vzor (Pattern)"

Podobně jako u řady dalších inženýrských termínů neexistuje žádná přesná definice tohoto pojmu. Mohli bychom snad napsat "popis opakovatelných řešení při návrhu struktury a chování softwarového produktu" nebo stručněji "řešení problému zařazené do daného kontextu". Kontextem se míní popis problému, který vzor řeší, zdůvodnění adekvátnosti řešení, doporučený postup při jeho implementaci, vztah k ostatním vzorům, atd. K úplnému vysvětlení a pochopení podstaty pojmu ale těchto několik slov zdaleka nestačí.

Vzor je více než opakovatelné řešení. Vzor je způsob popisu řešení. Vzor popisuje kontext, ve které je toto řešení použitelné, i to, jak konkrétní kontext ovlivní parametry řešení a jeho důsledky. Někdy může být vhodné popsat i špatné řešení, pokud chceme upozornit na často se opakující chybu.

Vzory pro návrh obsahují čtyři základní vlastnosti:

Jméno.

Jméno vzoru slouží k pojmenování daného řešení a jeho zařazení do katalogu vzorů.

Problém.

Popisuje možnost aplikace vzoru spolu s podmínkami nutnými pro to, aby vzor mohl být použit.

Řešení.

Nepopisuje konkrétní implementaci, protože vzor je určen pro použití v rozdílných situacích.

Použití.

Popisuje tradiční způsoby použití, jeho výhody a nevýhody.

3. Jak se vzory popisují

K této otázce existuje celá řada přístupů a je zřejmě ještě potřeba určitého času k tomu, aby se jednotlivé přístupy ověřily a na základě zkušenosti došlo k určitému sjednocení.

Mezi problémy, které se nejvíce diskutují patří:

- Mají být vzory popisovány spíše slovně nebo více strukturovaně s pomocí standardní šablony?
- Má se popis zaměřit na výslednou strukturu nebo spíše na proces jejího vytváření?
- Mají vzory popisovat již existující systémy (viz [6]) nebo poskytovat návod na konstrukci nových systémů (generativní vzory, viz [2])
- Mají být vzory pojmenovány jako instrukce nebo spíše výsledky aplikování instrukcí. Jedním slovem nebo větou.

Jediný konsensus, který v současné době v této oblasti existuje, je to, že musí být napsáno ještě hodně nových vzorů, než bude možné učinit podstatnější závěry.

Například Gamma a spol. viz [6] ve svém katalogu používají pro popis vzorů následující šablonu.

Záměr: Co vzor dělá. Co je jeho principem a účelem? Jakého konkrétního problému při návrhu se týká.

Motivace: Situace, ve které je vzor použitelný. Konkrétní otázka nebo problém, k němuž se vzor vztahuje a struktury objektů a tříd spojených s tímto problémem. Tato informace pomůže čtenáři pochopit abstraktní popis řešení, který vzápětí následuje.

Použitelnost: Ve kterých situacích lze vzor použít. Příklady špatného návrhu v situacích, ve kterých je vhodné vzor použít. Jak rozpoznat takové situace?

Zúčastněné objekty a třídy: Popis tříd a objektů obsažených v daném vzoru spolu se stanovením jejich úkolů

Spolupráce: Jak zúčastněné objekty spolupracují při vykonávání svých úkolů

Diagram: Grafická reprezentace vzoru.

Důležitost, závažnost, dosah: Jak vzor podporuje svou problémovou oblast. Jaké dohody jsou potřebné pro použití vzoru a jaké jsou výsledky jeho použití. Co vzor konkretizuje?

Implementace: Jakých omylů a postupů se vyvarovat při implementaci vzoru. Využití specifických vlastností daného programovacího jazyka.

Příklady:

Odkazy: Které vzory mají podobný účel. Jaké jsou hlavní rozdíly. S kterými jinými vzory by mohl být daný vzor použit.

Kent Beck Ve svém článku [2] pro zápis vzorů používá formát

Prekondice : Vzory, které musí být použity před aplikací tohoto vzoru

Problém: Problém řešený vzorem. Jeho popis slouží čtenáři pro rozhodnutí, zda je vzor požitelný v dané situaci

Konflikty: Typickým příkladem je kompromis mezi rychlostí vykonávání, potřebnou pamětí, složitostí programu atd.

Řešení : Stručný zápis řešení problému.

4. příklad

Následující příklad je převzat z [6]

Abstraktní továrna (Abstract Factory)

Záměr

Abstraktní továrna poskytuje rozhraní pro vytváření generických objektů (produktů). Odstraňuje závislost klienta, který objekty (produkty) vytváří na třídě popisující konkrétní produkt

Motivace

Uvažujme aplikaci, která podporuje různá uživatelská prostředí, např. Windows, Motif a Open Look, a která poskytují různé scroll bary pro každé z nich. Není vhodné zahrnout závislost na daném prostředí přímo do aplikace. Jeho volbu je vhodné odložit na dobu běhu programu. Zavedení třídy ScrollBar omezuje flexibilitu a znovupoužitelnost vynucenou vazbou na konkrétní třídu místo na daný protokol. Abstraktní továrna umožní vyhnout se této vazbě. Abstraktní třída WindowKit deklaruje služby pro vytvoření scroll baru a dalších ovladačů (např. CreateScrollBar()). Ovladače pro Windows, Motif i Open Look jsou odvozeny z této společné třídy. Klient přistupuje k danému ovladači prostřednictvím interface k WindowsKit.

Použitelnost

V případech, kdy vytvářené objekty mohou být různých tříd a aplikace nemá být závislá na těchto třídách. Když různost při vytváření, skládání nebo reprezentování agregovaných objektů nebo subsystémů nesmí být zahrnuta do aplikace. Klient sám explicitně nevytváří takové objekty, ale předá tuto činnost třídě AbstractFactory. Volá tedy metodu třídy AbstractFactory a ta mu vrátí objekt poskytující přístup k požadované struktuře.

Zúčastněné objekty a třídy

AbstractFactory
deklaruje obecný interface pro operace, které vytváří generické objekty.

ConcreteFactory
definuje operace, které vytváří konkrétní objekty.

GenericProduct
definuje generický interface pro vytvářené objekty(produkty).

SpecificProduct
definuje konkrétní objekt(produkt) vytvořený příslušnou ConcreteFactory.

Spolupráce

Obvykle je při běhu programu vytvořena jedna instance ConcreteFactory. Ta produkuje objekty mající konkrétní implementaci. Pro produkci jiných objektů musí být klienti překonfigurováni pro použití jiné ConcreteFactory. AbstractFactory přenese vytvoření produktu na svou podtřídu ConcreteFactory.

Důležitost, závažnost, dosah

Zaměřuje se při vývoji na možnost změny typů objektů vytvářených klientem. Izoluje klienta od implementace tříd. AbstractFactory, jež zahrnuje jméno třídy do signatury operací může být obtížně rozšiřitelná o nové druhy produkovaných objektů. Většinou je pak nutné předeklarování AbstractFactory i všech ConcreteFactory. AbstractFactory může být složena z podřízených factory objektů. Zodpovědnost za vytvoření objektů je delegována na tyto sub-factory. Kompozice abstraktních továren poskytuje jednoduchý způsob jak doplnit nové druhy objektů, za jejichž vytvoření je továrna zodpovědná.

Příklady

V ET++ je pomocí AbstractFactory dosaženo přenositelnosti mezi různými window systémy. Abstraktní třída WindowSystem definuje rozhraní pro vytváření objektů reprezentujících zdroje systému (MakeFont, MakeColor,...) Konkrétní podtřídy implementují interface pro specifický window systém. Při běhu programu se vytváří instance konkretní podtřídy pro daný systém.

Implementace (Implementation)

Neobvyklá implementace je možná ve Smalltalku. Protože třídy jsou zde objekty, není nutné mít různé ConcreteFactory pro vytváření různých produktů. Je možné uložit třídy v proměnných. Tyto třídy vytváří nové instance v zastoupení konkrétní továrny. Pouze třídy uložené v proměnných se pak zamění.

5. Jaký přínos lze od vzorů očekávat.

Katalog vzorů lze využívat zejména pro:

1. Řešeni problémů: příručka vzorů obsahuje seznam prozkoumaných a ověřených řešení spolu s návodem jak a kdy je aplikovat. Používání vzorů tímto způsobem může zdokonalit a zefektivnit řešení problémů a usnadnit pochopení důsledků, které zvolené rozhodnutí bude mít.

2. Výměna znalostí: Existuje konsensus v tom, že vzory slouží pedevším pro komunikaci mezi lidmi (na rozdíl od komunikace člověk-počítač, ke které jsou určeny programovací jazyky). Katalog vzorů j prostředkem pro uchovávání know-how organizace ve formě, která není závislá na jednotlivých programátorech.

3. Dokumentace systému: Existuje-li katalog vzorů, není třeba opakovat jejich popis v dokumentaci. Dokumentace se se pak může zaměřit pouze na specifika daného systému.

4. Zjednodušení struktury programu: Jsou-li některé vzory důsledně zahrnuty do architektury systému, pak to znamená, že nestálost systému je zmenšena a jeho struktura obsahuje některé invarianty.

5. Závěr

Kniha E. Gamma, R. Helm, R. Johnson, J. Vlissides viz [6] obsahuje rozsáhlý katalog vzorů. Další informace o vzorech je možno získat v konferenci patterns-request@cs.uiuc.edu.

Počet knih zabývajících se "pattern programming" velmi rychle roste.

Literatura.

[1] G. Booch. Object-Oriented Analysis and Design with Applications. Benjamin/Cummings, Redwood City, CA 1994. Second Edition.

[2] K. Beck. "Patterns and Software Design" Dr. Dobb's Journal Feb. 1994

[3] J. Coplien. "Pattern Languages for Organizational Process" Object Magazine July/Aug 1994

[4] J. Coplien. "Software Design: The Emerging Patterns" C++ Report July/Aug 1994

[5] J. Coplien. "Examining the Software Development Process" Dr. Dobb's Journal Oct. 1994

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Addison Wesley, 1995.

V.Snášel, V.Sklenář
Katedra matematické informatiky
Universita Palackého Olomouc
TOMKOVA 40
OLOMOUC
779 00
fax 068 5411643
tel 068 5411642
e-mail: snasel@risc.upol.cz