Návrhové vzory GoF

© 2008 Radek Ošlejšek
FI MU Brno

oslejsek@fi.muni.cz
http://www.fi.muni.cz/~oslejsek/PA103

Návrhové vzory

The Gang-of-Four: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Design Patterns: Elements of Reusable Object Oriented Software (1995).

Vlissidies pronunciation (dialogue from the internet discussion with his long-time colleague):

Q: And I'd like to make sure I pronounce his name correctly.

A: The weird part is I've known John for years and I'm not even sure :-)

The pronunciation I generally use is VLIH-Suh-dees. I've heard others use VLIH-SEE-DEES...

Pozadí

- Návrh znovupoužitelného softwaru je obtížný
  - nalezení vhodných objektů a abstrakcí
  - pružnost, modularita, elegance $\Rightarrow$znovupoužití
  - trvá delší dobu, než se objeví, cesta pokusů a omylů
- Úspěšné návrhy přesto existují
  - vykazují opakující se třídy a struktury
- Jak popsat tyto opakující se struktury ?

Původ vzorů

- architekt Christopher Alexander
  (pojem „vzor" cca 1977-1979)
- Kent Beck and Ward Cunningham, Textronix, OOPSLA'87
  (použil Alexanderovy ideje o „vzorech" pro návrh Smalltalk GUI)
- Erich Gamma, Ph.D. teze, 1988-1991
- James Coplien, *Advanced C++ Idioms book*, 1989-1991
- Gamma, Helm, Johnson, Vlissides ("Gang of Four“ - GoF) *Design Patterns: Elements of Reusable Object-Oriented Software*, 1991-1994
- PLoP konference a knihy, 1994-....
- Buschmann, Meunier, Rohnert, Sommerland, Stal, *Pattern – Oriented Software Architecture: A System of Patterns* (“POSA”)

Ch. Alexander - jazyk vzorů

- Co je to, co dává budově *Kvalitu*?
  - svoboda, život, pohodlí, harmonie
- Vzor: řešení problému v daném kontextu
  - *Vstupní přechod*
  - *Gradient soukromí*
  - *Světlo na dvou stranách každého pokoje*
- Propojené vzory $\Rightarrow$ *Jazyk vzorů*
  - 2534 vzorů, hrubé až po jemnozrnné

The Timeless Way of Building (1979)
A Pattern Language (1977)

Návrhový vzor

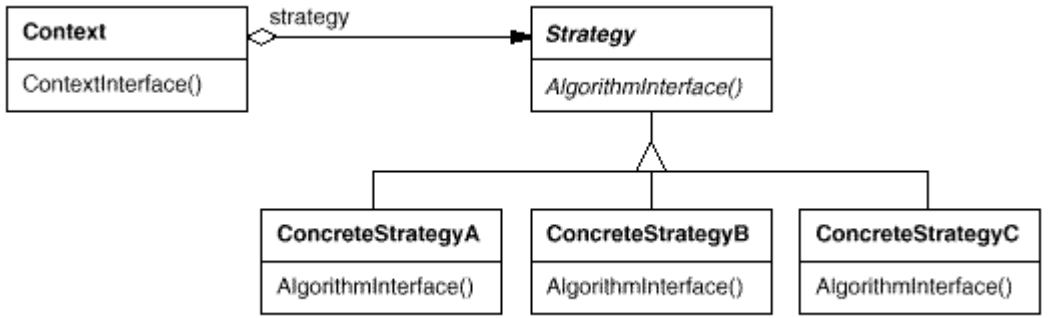
- Popisuje opakovaně se objevující návrhovou strukturu
  - abstrahuje od konkrétních návrhů
  - identifikuje třídy, spolupráce, zodpovědnosti
  - aplikovatelnost, kompromisy, důsledky
- Vzory nejsou návrhy
  - Metamodely, které musí být instanciovány a současně je nutné
    - vyhodnotit kompromisy a důsledky
    - učinit návrhová a implementační rozhodnutí
    - implementovat, kombinovat s jinými vzory
- Společný slovník pro návrh
  - “tady použijeme vzor Observer”
    - zvýšená rychlost návrhu, kultura
  - sdílený slovník
    - Uvnitř/mezi týmy, řízení na různých úrovních
- Příklady
  - Strategie: algoritmy jako objekty
  - Composite: rekurzivní struktury

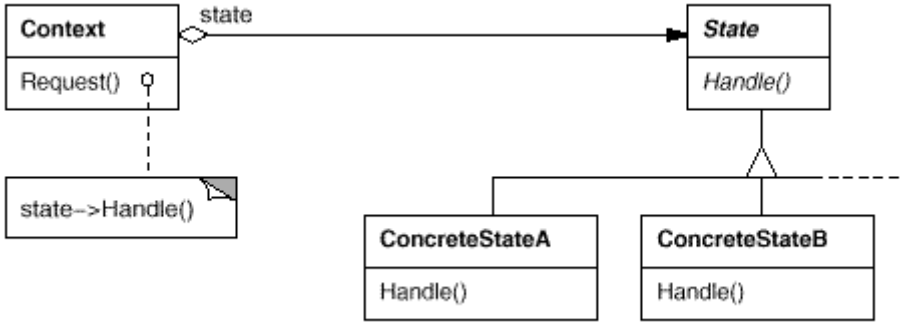
Vzory vs. důvěra v návrh

- obecná nezkušenost s objekty
  - je můj návrh v pořádku?
- vzory rodí důvěru
  - vždy můžete svést vinu na “Gang of Four”
  - ponechávají prostor pro kreativitu
- většina lidí zná vzory
  - částečně, bez plného pochopení,
  - připouštějí, že ostatní mají podobné návrhy
  - vzory se vylepšují s jejich používáním

Společné problémy

- Krocení přílišného nadšení
  - „vítězíte“, pokud jste použili většinu vzorů
  - řešení nesprávného problému
  - související výdaje a cena
  - vše řeší poslední vzor, který jste se právě naučili
- Struktura místo cíle
  - všechno je Strategie
  - vzory používají podobné konstrukce

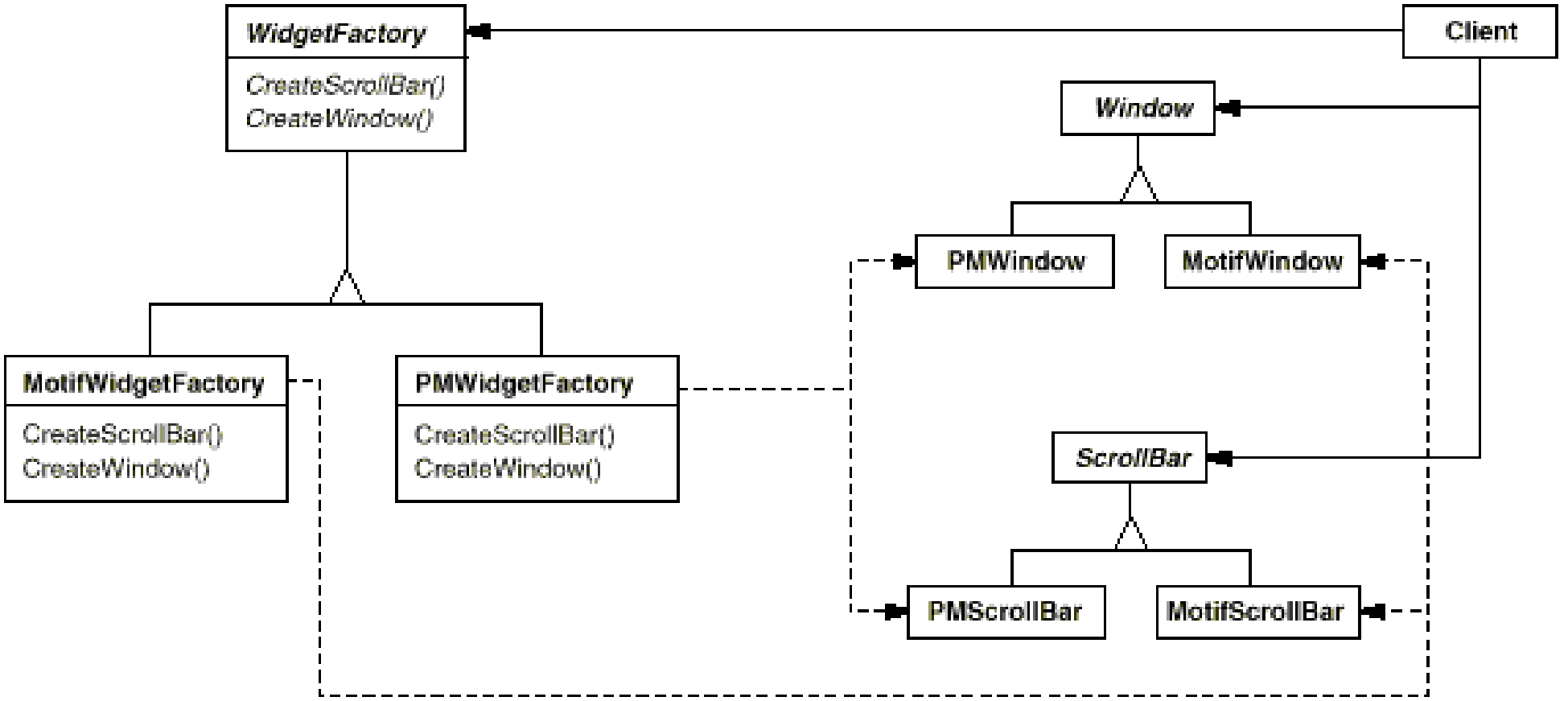




Typy GoF vzorů

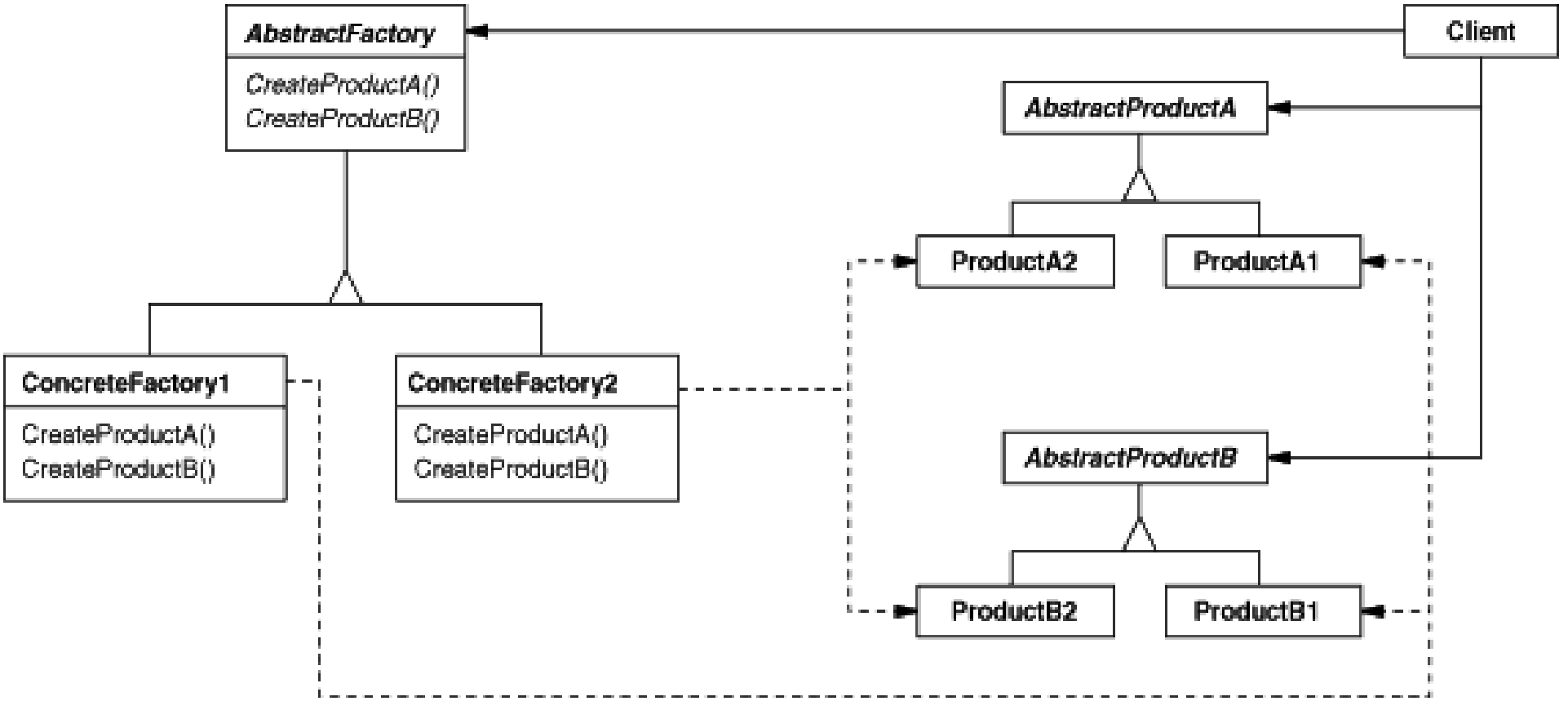
- Tvořící vzory (*creational p.*):
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Strukturální vzory (*structural p.*):
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- Vzory chování (*behavioral p.*):
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

Abstract Factory - motivace



Motif = GUI pro UNIX
PM = Presentation Manager = GUI pro OS/2

Abstract Factory - vzor

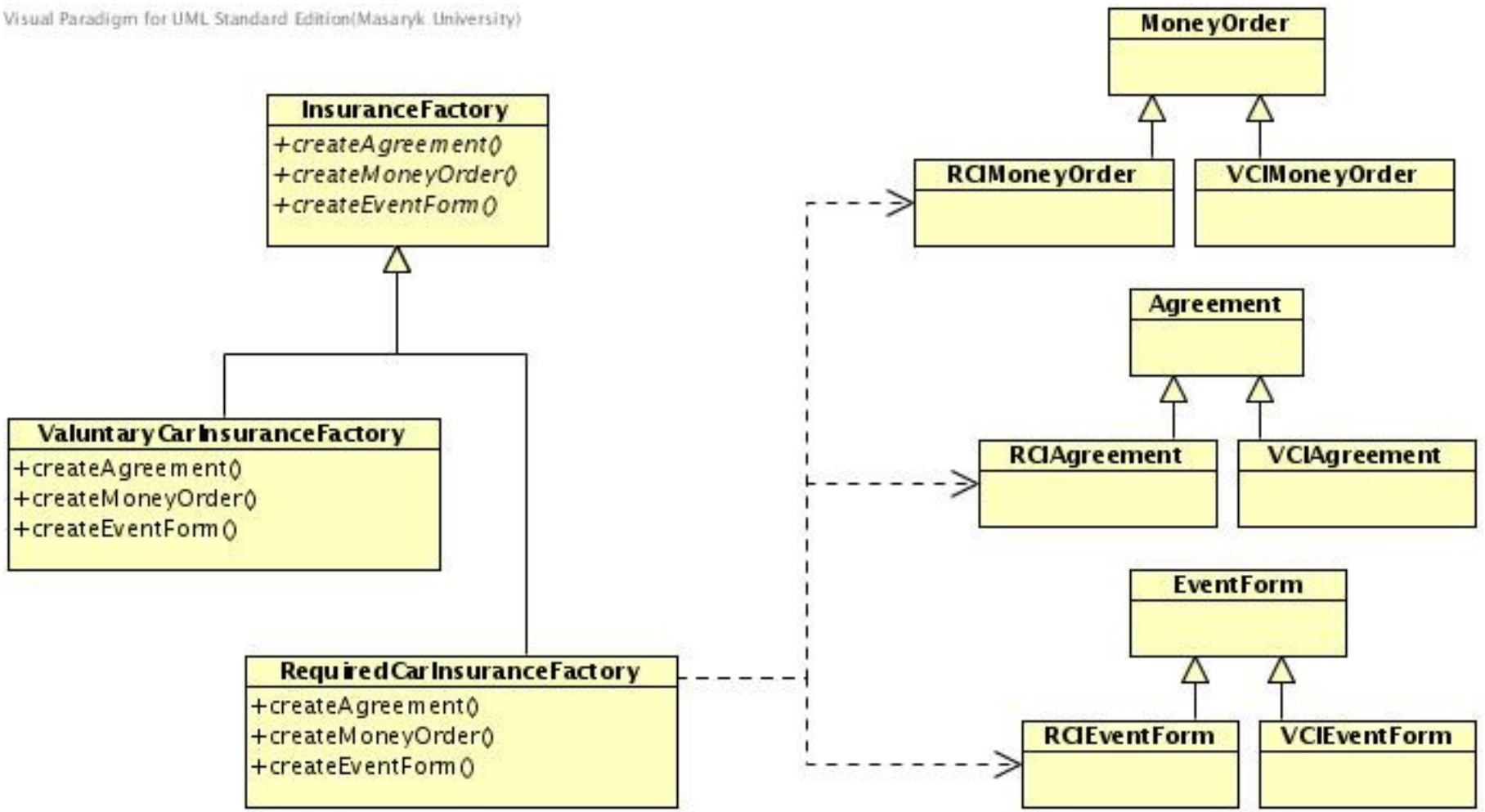


Abstract Factory - charakteristiky

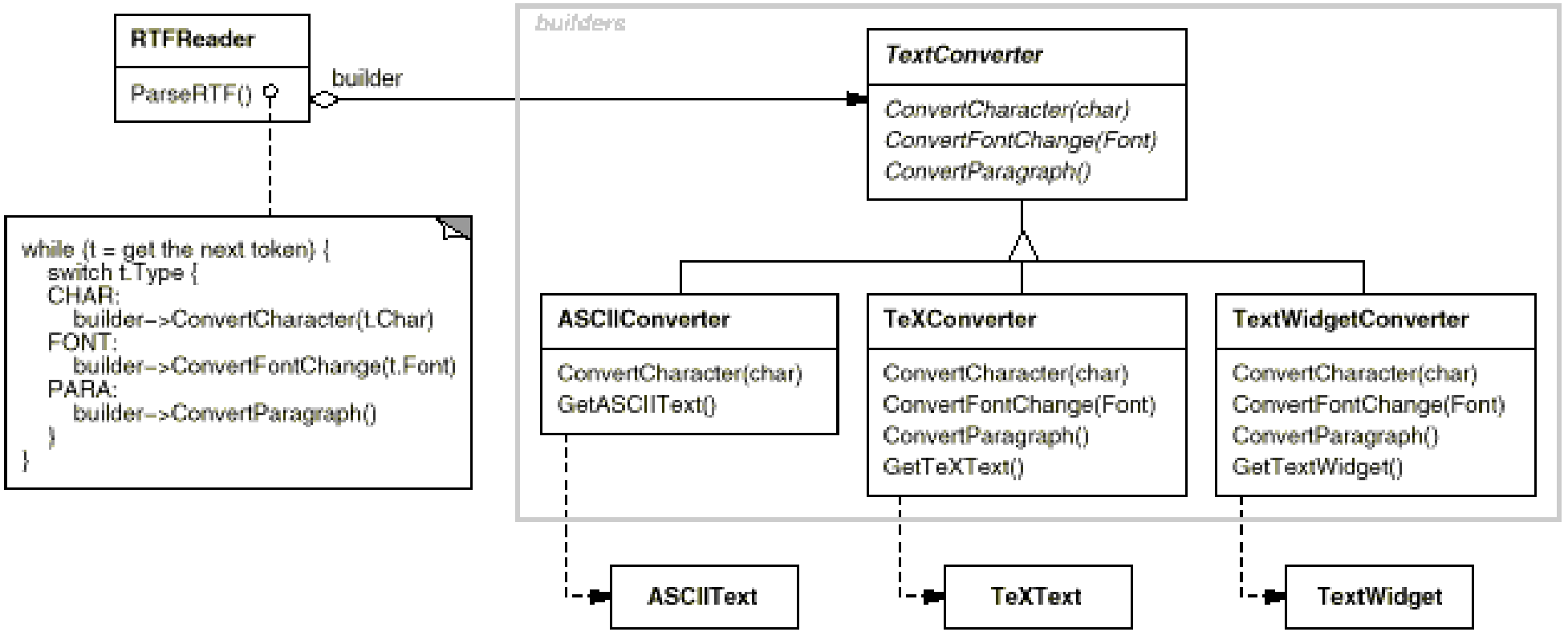
- Aplikovatelnost
  - Systém nemá být závislý na tom, jak jsou výrobky vytvářeny, sestavovány a reprezentovány
  - Systém bude konfigurován pro jednu z mnoha rodin výrobků
  - Rodina souvisejících výrobků bude používána společně a toto je potřeba zajistit (vynutit)
  - Vytváříme objektovou knihovnu výrobků a chceme zveřejnit pouze jejich rozhraní, ne implementaci
- Důsledky
  - Izolace konkrétních tříd
    - produkty jsou implementovány vně klienta, ten používá jen obecné rozhraní.
  - Snadná záměna rodin výrobků
  - Podpora pro konzistenci výrobků
  - Podpora nových druhů výrobků je obtížná.

Abstract Factory - pojišťovna

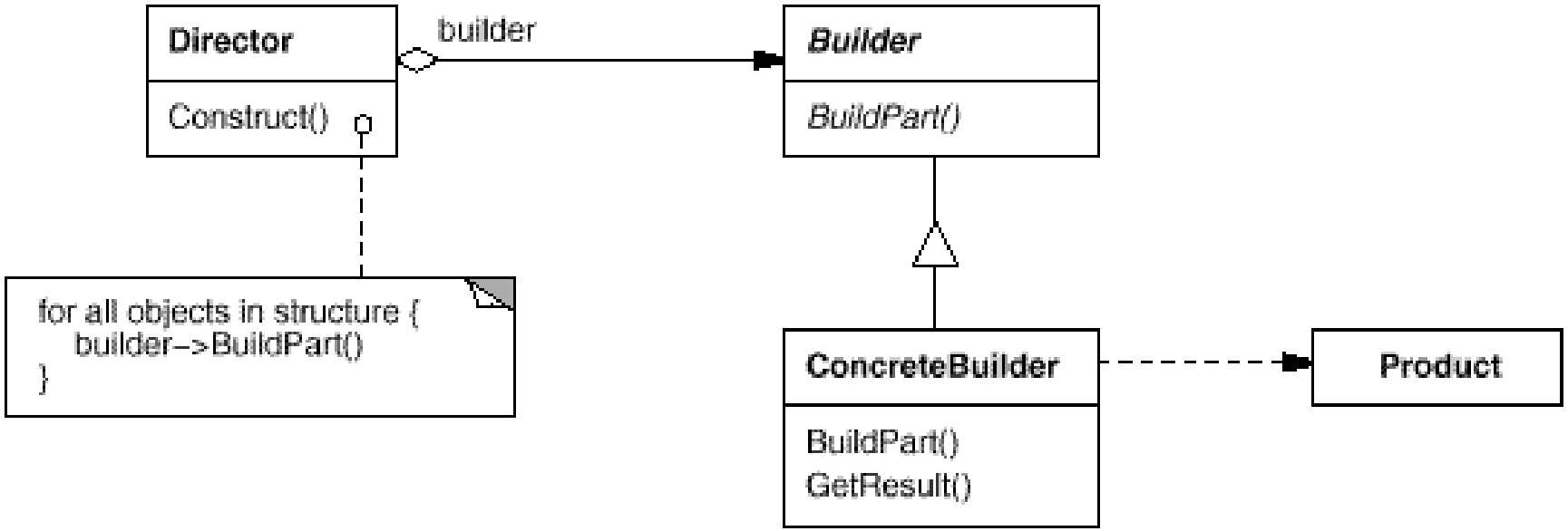
Příklad



Builder - motivace

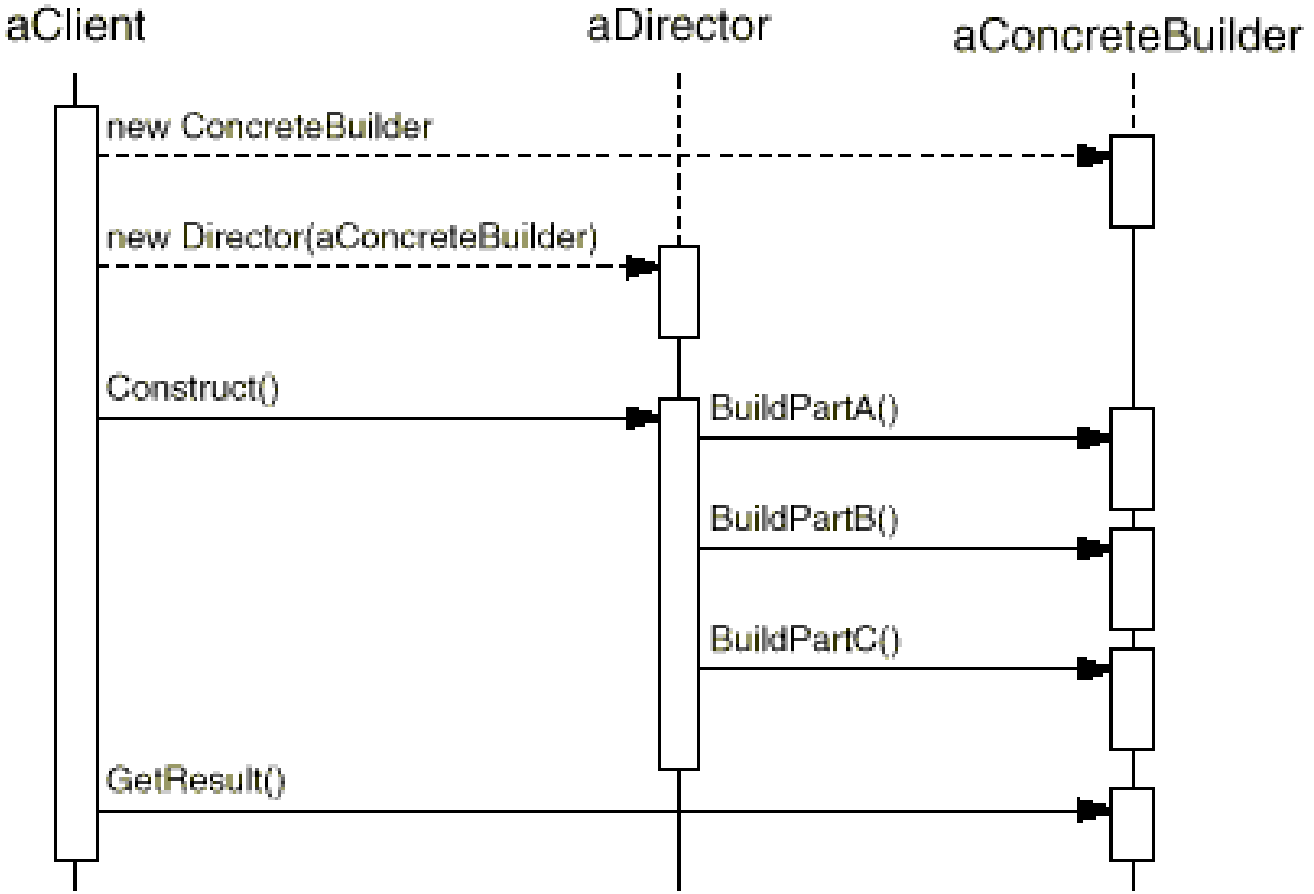


Builder - vzor



Builder

spolupráce

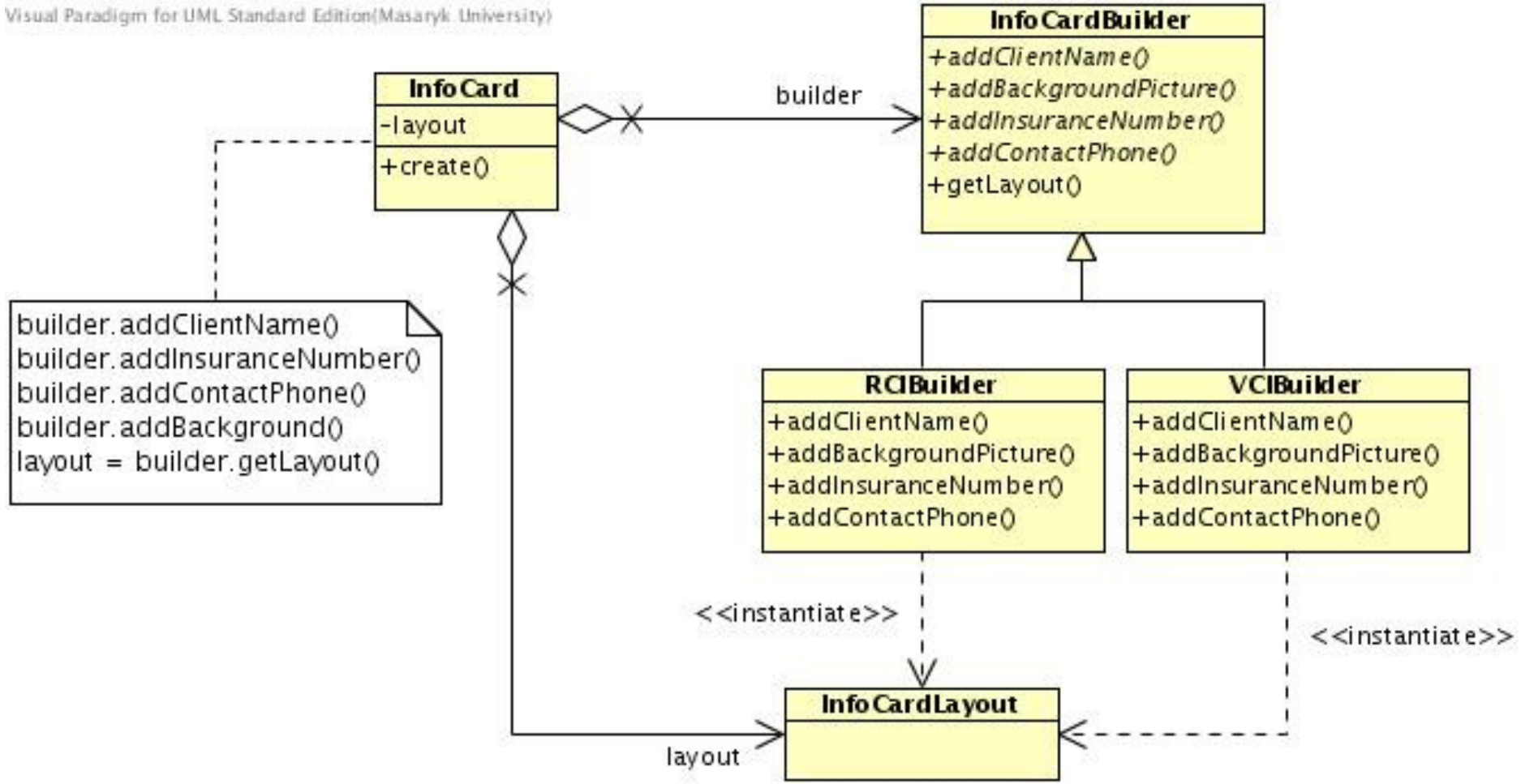


Builder - charakteristiky

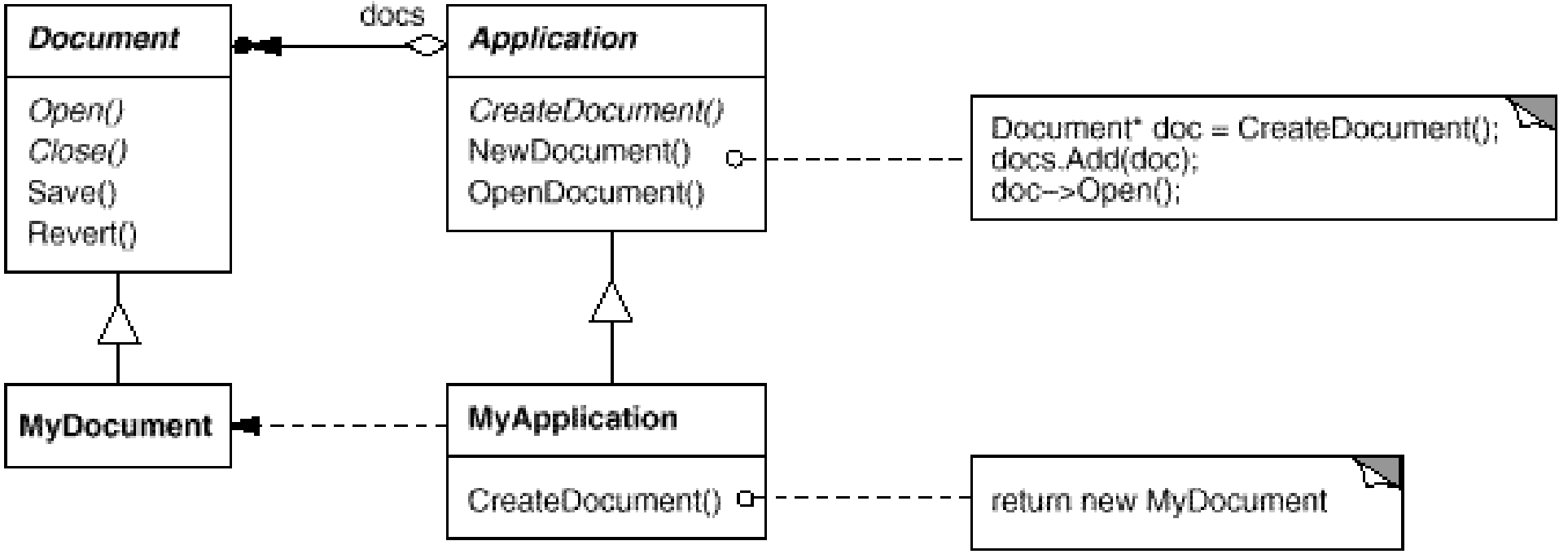
- Aplikovatelnost
  - Algoritmus pro tvorbu složitého objektu by neměl záviset na částech, které tvoří objekt, a na tom, jak jsou sestaveny
  - Konstrukční proces musí umožnit různé reprezentace konstruovaného objektu
- Důsledky
  - Je možné měnit interní reprezentaci produktu
  - Kód pro konstrukci a reprezentaci produktu je izolován
    - => *SGMLReader* používající konvertory z příkladu pro převod SGML dokumentů
  - Poskytuje jemnější kontrolu konstrukčního procesu
    - => vhodné zejména pro produkty se složitější strukturou

Builder - pojišťovna

Příklad

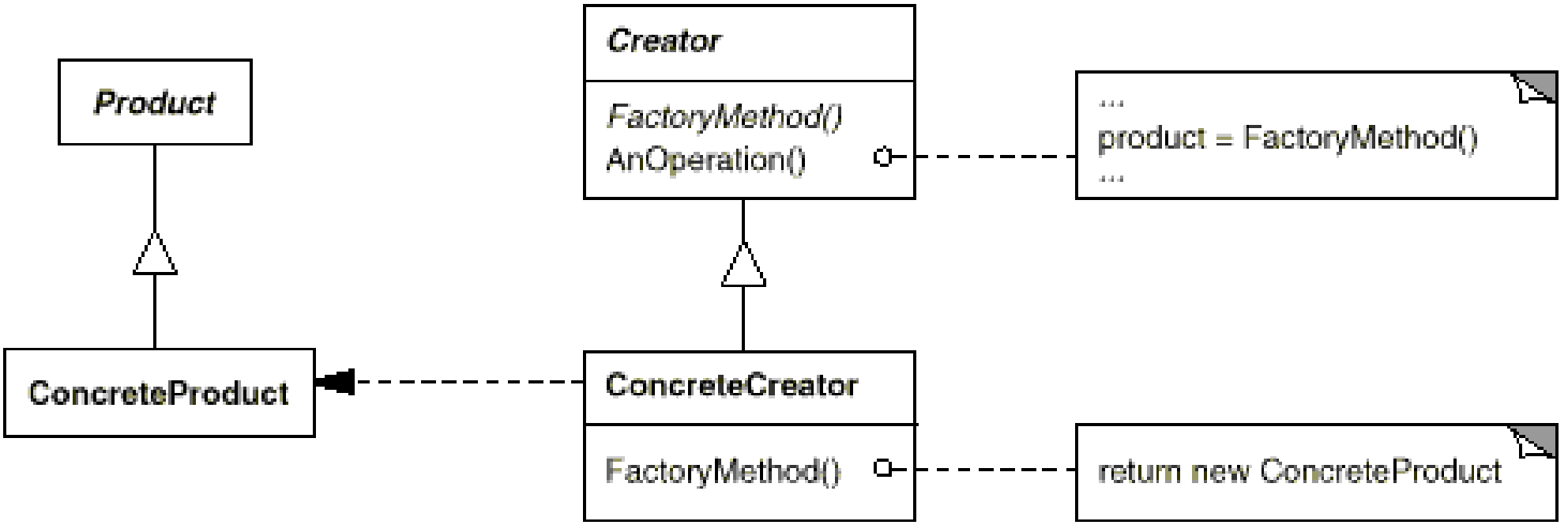


Factory Method - motivace



Aplikace ví, ***kdy*** má být dokument vytvořen, neví ale, ***jaký druh*** dokumentu se má vytvořit.

Factory Method - vzor

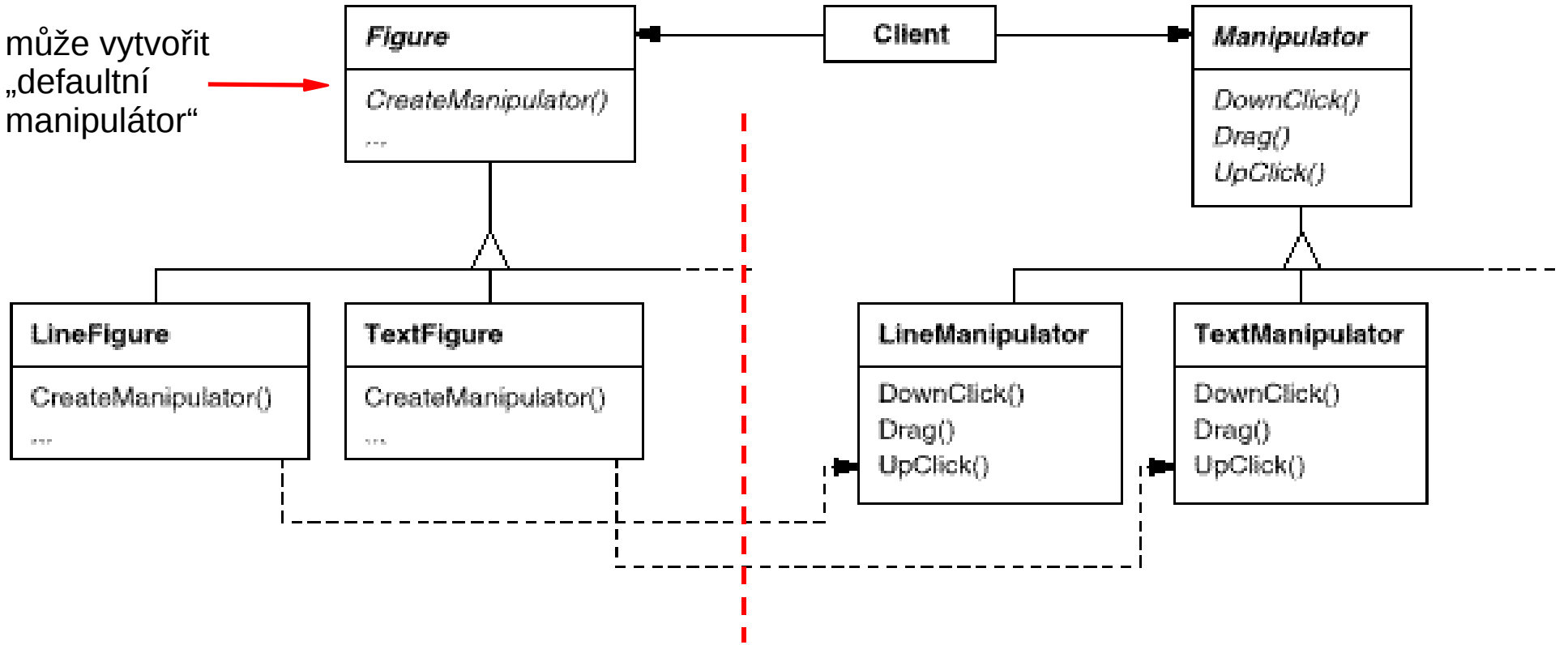


Factory Method - charakteristiky

- Aplikovatelnost
  - Třída nemůže předvídat třídu objektů, které musí vytvářet
  - Třída požaduje, aby její podtřídy specifikovaly vytvářené objekty
  - Třídy delegují zodpovědnost na jednu z několika pomocných podtříd a znalost této delegace chceme lokalizovat
- Důsledky
  - Poskytuje záchytná místa („*hooks*“) pro podtřídy
  - Propojuje paralelní hierarchie tříd

Příklad paralelní hierarchie

Motivace pro vytvoření paralelní hierarchie:
- Informace nutné pro manipulaci s obrázky jsou platné pouze v čase samotné manipulace.
- Není vhodné je proto ukládat přímo do obrázku, ale je lepší použít speciální třídu s touto zodpovědností.

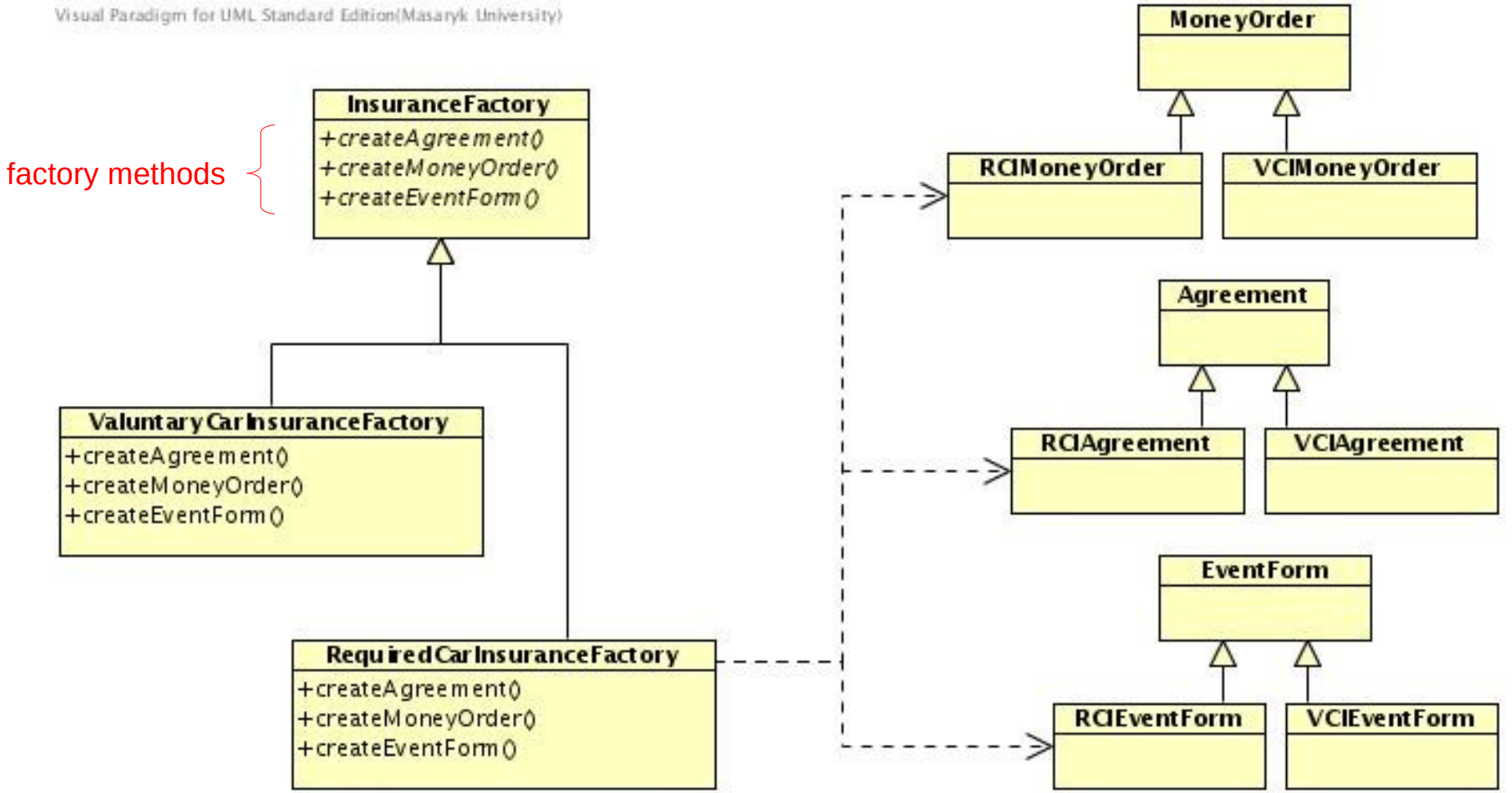


Paralelní hierarchie = třída deleguje část své zodpovědnosti do separátní třídy

Factory Method - pojišťovna

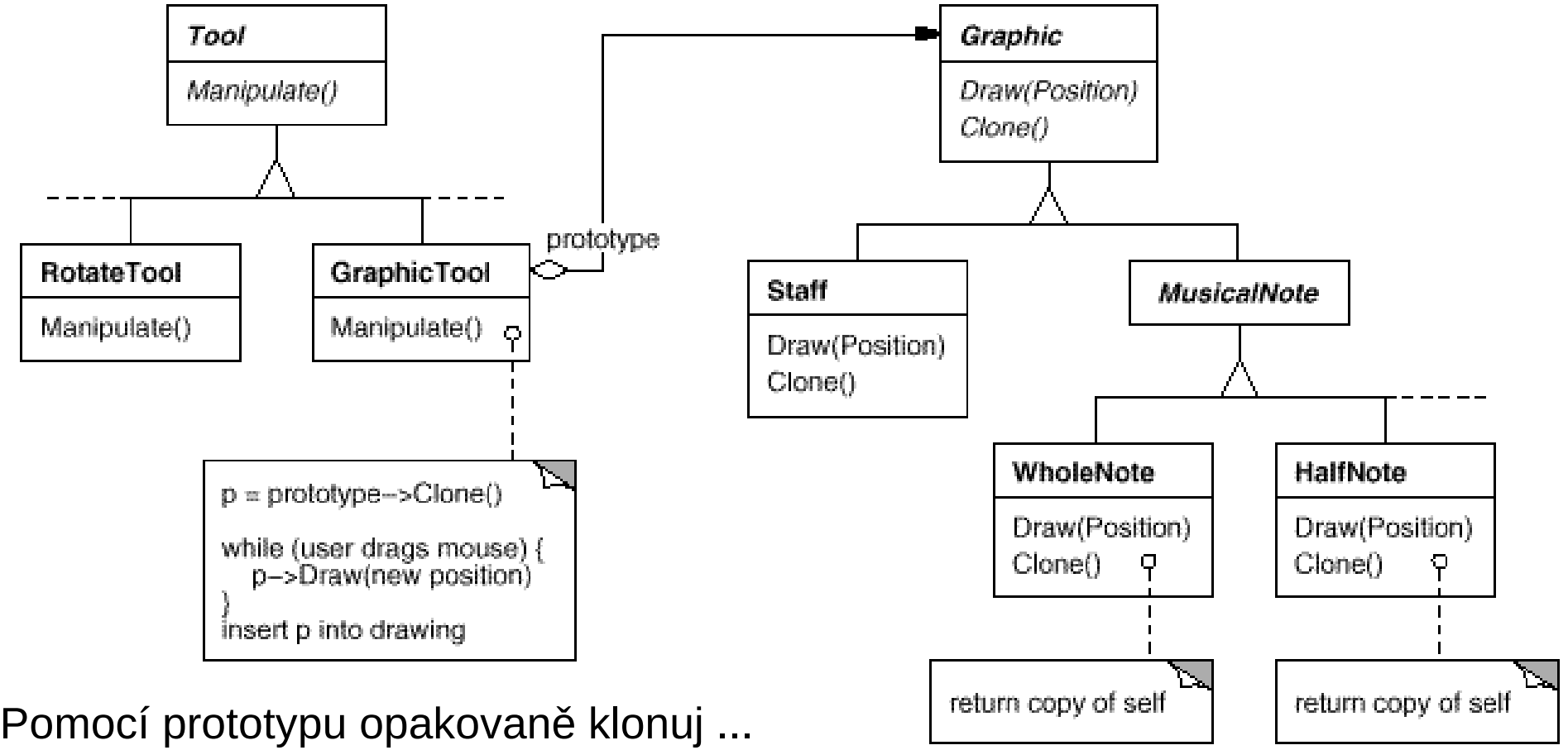
Příklad

Viz Abstract Factory:



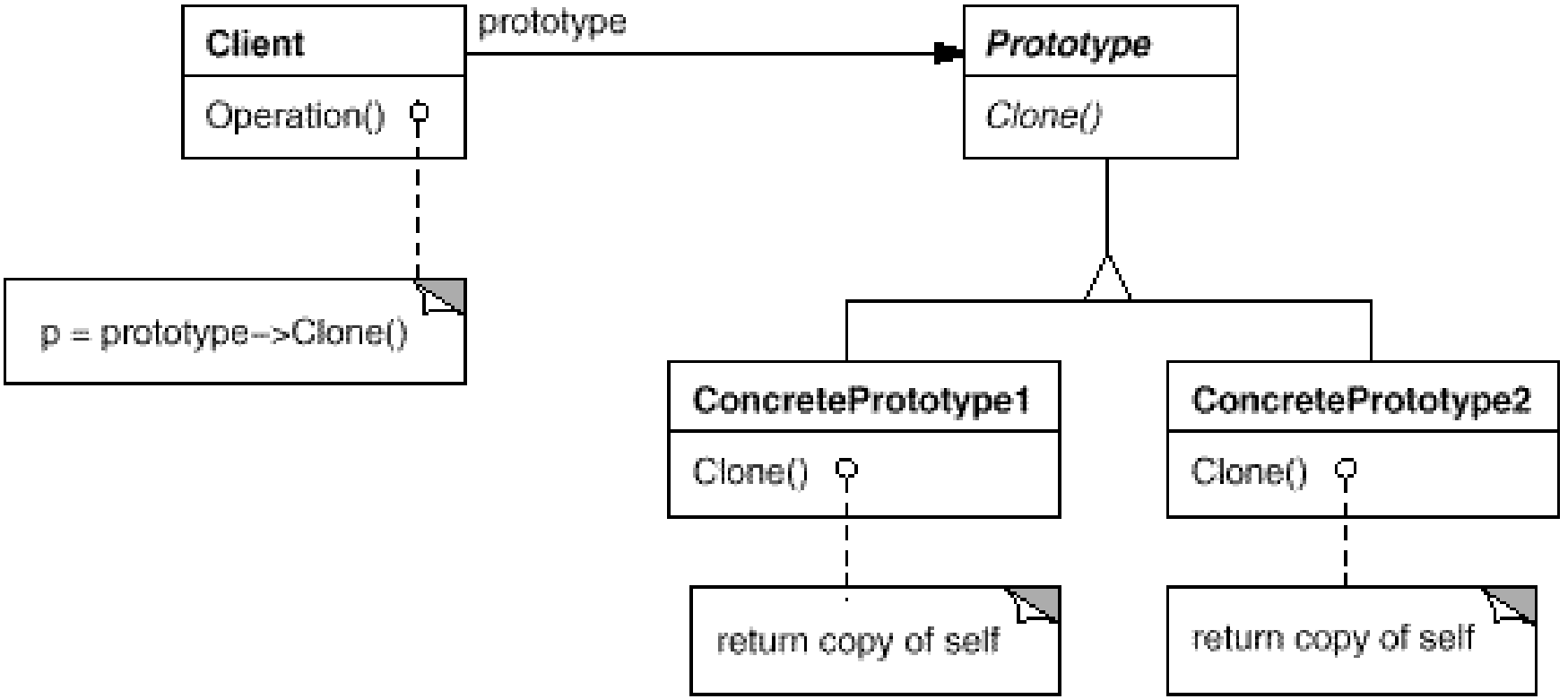
Prototype - motivace

Motivace:
- Rozšiřujeme existující framework pro práci s grafickými prvky o práci s notami.
- Různé nástroje slouží pro různé typy manipulací s grafickými prvky.
- GraphicTool slouží na vytváření nových grafických prvků na obrazovce.
- Jak GraphicTool pozná, kterou instanci vytvořit?
  - Mnoho instancí GraphicTool s různými prototypy grafických prvků + klonování



Pomocí prototypu opakovaně klonuj ...

Prototype - vzor

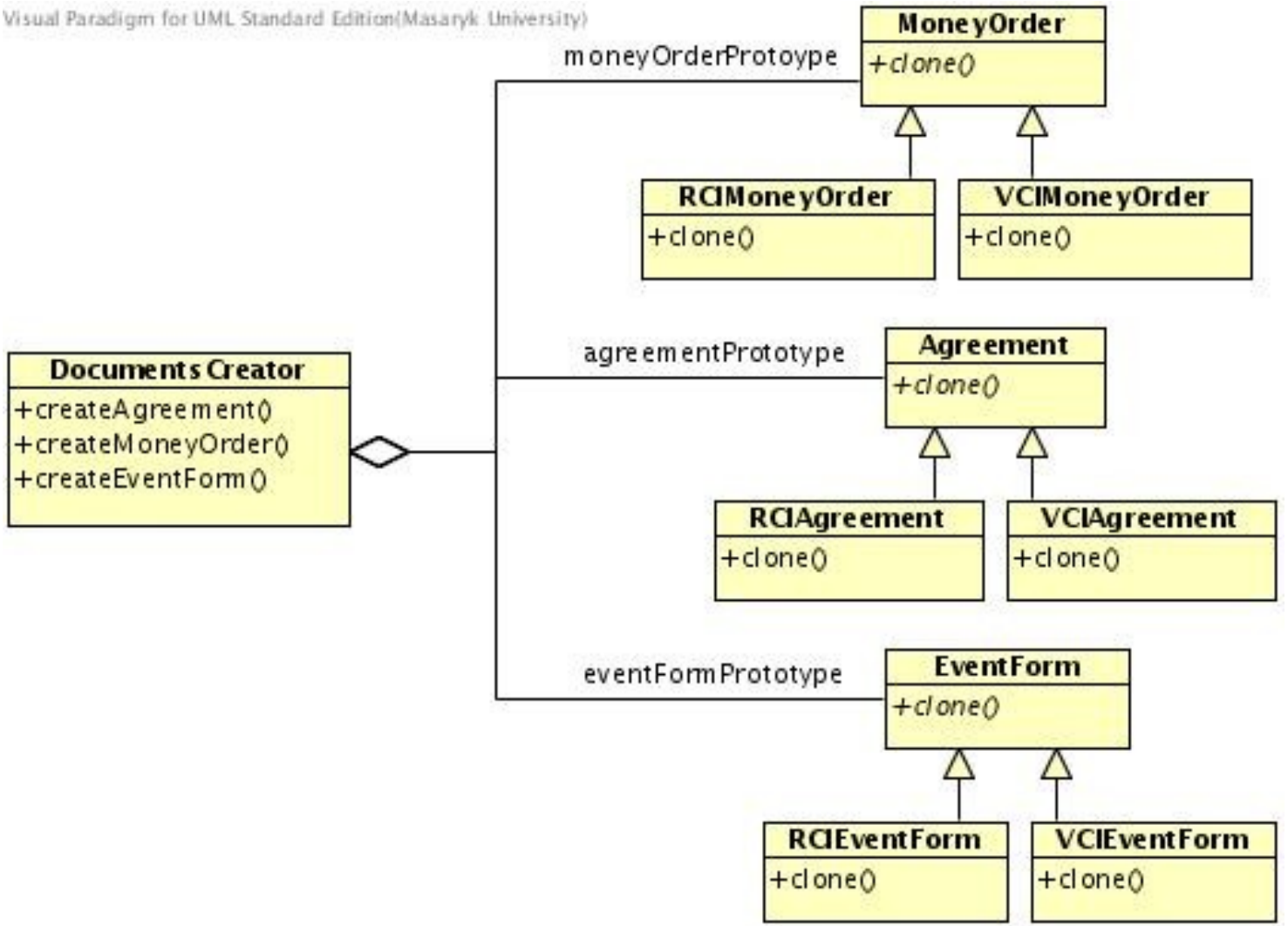


Prototype - charakteristiky

- Aplikovatelnost
  - Systém by měl být nezávislý na způsobu tvorby, skládání a reprezentace objektů *a zároveň*
    - pokud jsou instanciované třídy specifikovány za běhu programu, *nebo*
    - chceme se vyhnout tvorbě hierarchií továren, které jsou paralelní k hierarchiím výrobků, *nebo*
    - instance třídy mají jeden stav z několika málo kombinací.
- Důsledky
  - Umožněno přidávání a rušení výrobků za běhu.
  - Specifikace nových objektů pomocí proměnných hodnot.
    - prototypy jednoho objektu v různých stavech
  - Specifikace nových objektů pomocí proměnné struktury.
    - zkopírování složité struktury namísto jejího vytváření
  - Redukce podtříd.
  - Dynamická konfigurace aplikace s třídami.

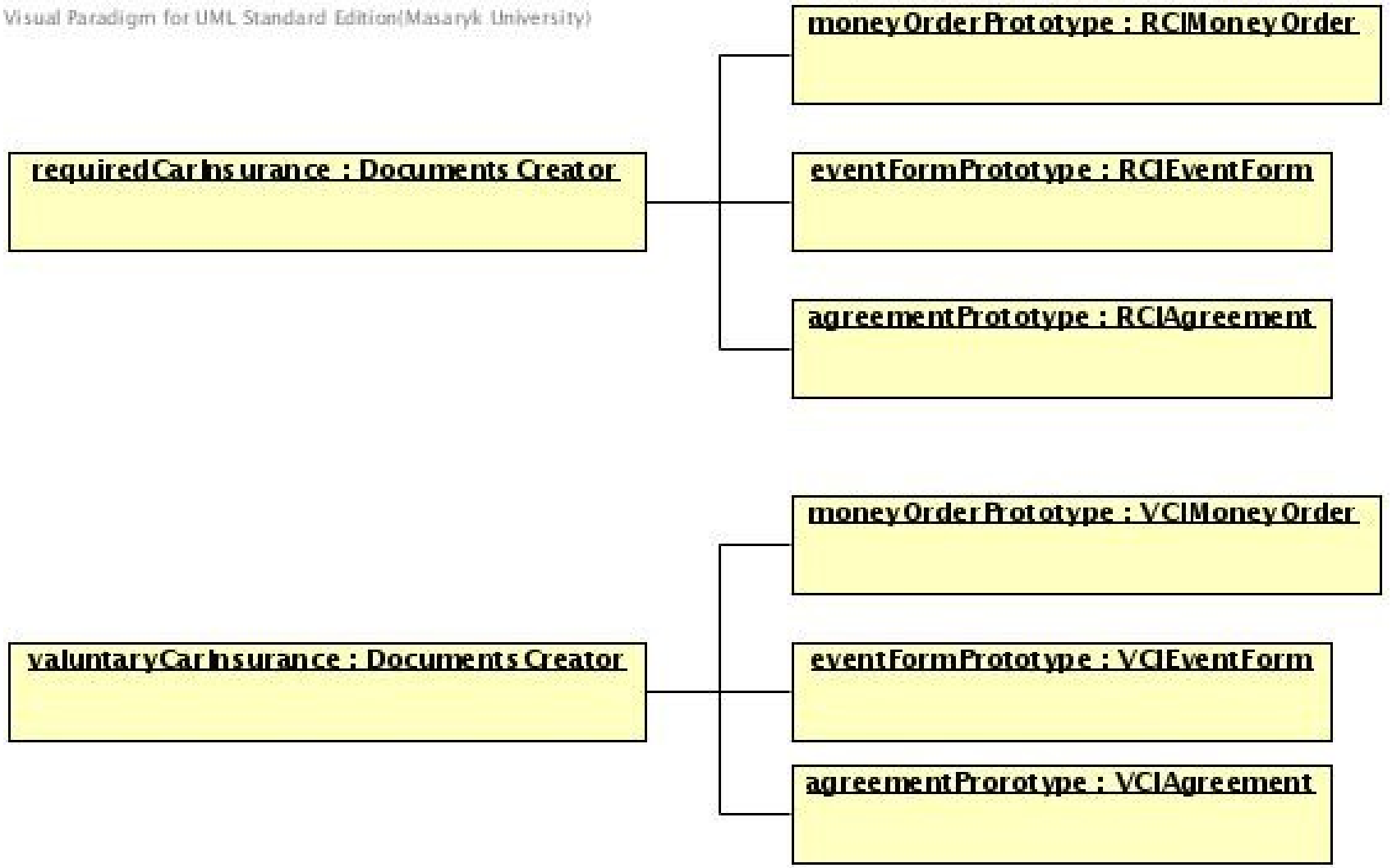
Prototype - pojišťovna

Příklad

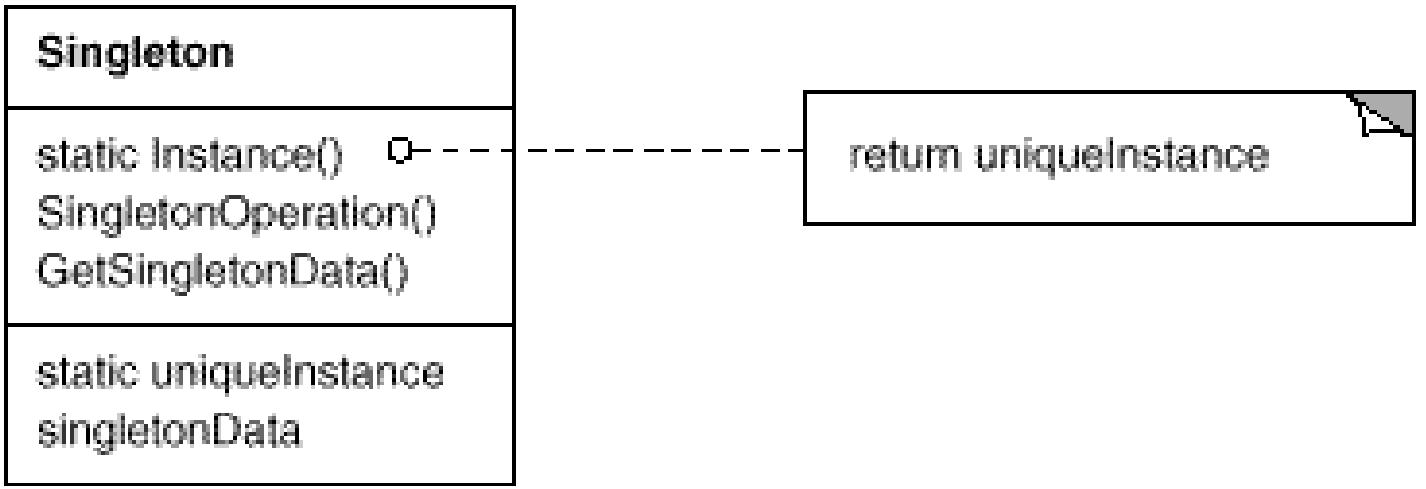


Prototype – pojišťovna

Příklad



Singleton - vzor



Singleton - charakteristiky

- Aplikovatelnost
  - Musí existovat právě jedna instance třídy a musí být přístupná klientům ve známém přístupovém bodě
  - Instance má být rozšiřitelná mechanismem podtříd a klienti by měli mít možnost používat rozšířené instance bez modifikace vlastního kódu
- Důsledky
  - Řízený přístup k jediné instanci.
  - Redukce jmenného prostoru
    - Náhrada mnoha globálních proměnných jedním singletonem
  - Povoluje úpravu operací
    - Chování singletonu může být změněno jeho podtřídou. Konfigurace aplikace pro použití podtřídy může být dokonce provedena za běhu.
  - Povoluje změnu počtu instancí (úprava vzoru)
  - Pružnější, než pomocí operací třídy
    - Téhož chování lze dosáhnout pomocí statických metod, u kterých ale obvykle chybí polymorfismus. Stejně tak je obtížné pracovat s více instancemi.

Singleton - pojišťovna

Příklad