Service Access and Configuration Patterns

Service Access and Configuration Patterns

- WrapperFacade
- Component Configurator
- Interceptor
- Extension Interface

Service Access and Configuration Patterns

- **Vlastnosti, cíle**
  - Dynamická změna funkcionality bez rekompilace
  - Snadné přidávání, odebírání a konfigurace komponent
  - Zvýšení přenositelnosti
  - Odstínění od networkingu

Wrapper Facade - Cíle

- **Zvýšit úroveň abstrakce**
- **Funkce převést na třídy a rozhraní**
- **Jednotné rozhraní na všech platformách**
- **=> Zapouzdřit práci s vlákny, zámky, sokety, GUI..**

Wrapper Facade

- **Facade**
  - Zakrývá komplexní vztahy mezi objekty
- **Wrapper Facade**
  - Zakrývá low-level rozhraní
- **Low-level API**
  - Nízká úroveň abstrakce (nebývá objektové)
  - Nekompatibilita mezi platformami
  - Náročné na použití
  - Náchylné na chyby

Wrapper Facade – Jak na to

- **Seskupit funkce pracující se stejnými datovými strukturami**
- **Identifikovat průnik funkcionalit na podporovaných platformách**
- **Skupiny + průniky = třídy wrapper fasády**
- **Ponechat přístup k nízkorúrovňovým datovým strukturám (handle, ukazatel)**

Wrapper Facade – Příklad

```
class Thread_Mutex
{
public:
  Thread_Mutex (void) {
    InitializeCriticalSection (&mutex_);
  }

  ~Thread_Mutex (void) {
    DeleteCriticalSection (&mutex_);
  }

  int acquire (void) {
    EnterCriticalSection (&mutex_); return 0;
  }

  int release (void) {
    LeaveCriticalSection (&mutex_); return 0;
  }

private:
  // Win32-specific Mutex mechanism.
  CRITICAL_SECTION mutex_;

  // = Disallow copying and assignment.
  Thread_Mutex (const Thread_Mutex &);
  void operator= (const Thread_Mutex &);
};
```

Wrapper Facade – v praxi

- **Použití v praxi**
  - MFC, OpenGL, Qt, WPF

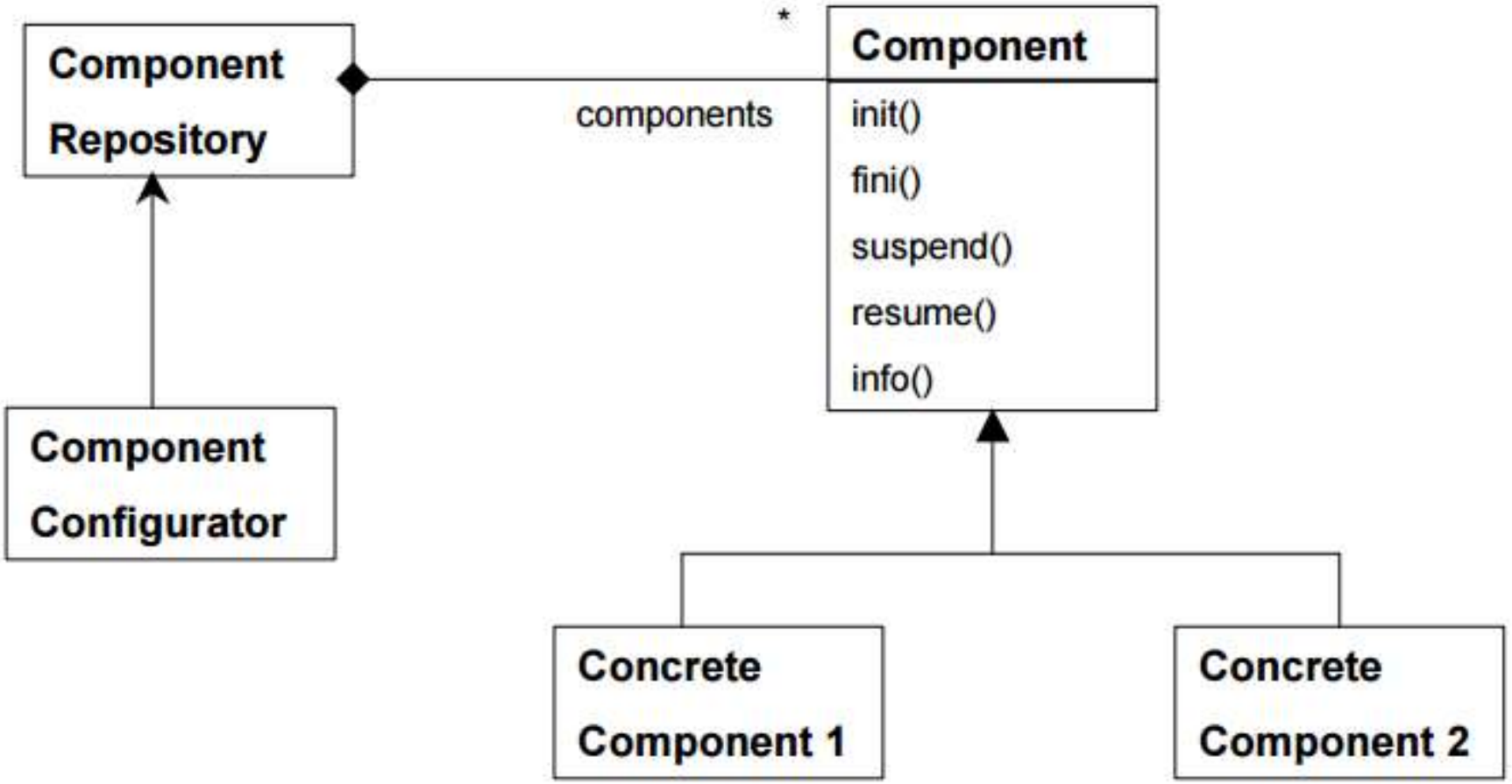
Component Configurator - motivace

- **Změna implementace znamená restart celého programu**
  - Co když program musí běžet 24/7 ?
- **Program běží v různých prostředích**
  - V každém potřebuje jinou konfiguraci

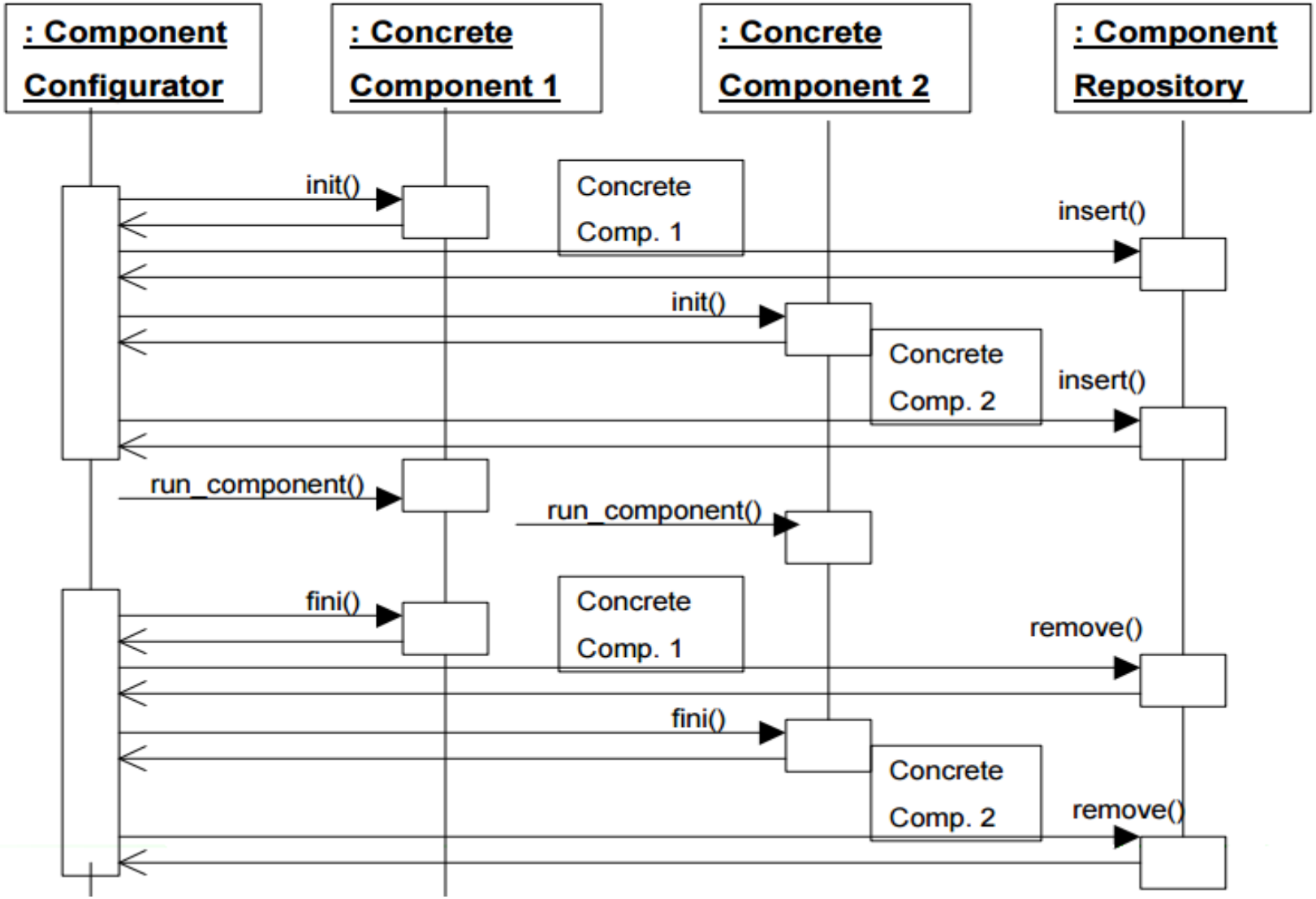
Component Configurator

- **Životní cyklus komponent**
  - Inicializace, zastavení, opětovné spuštění, terminace
- **Component configurator které se mohou za běhu měnit**
  - Spravuje komponenty a jejich životní cykly
- **Component repository**
  - Udržuje seznam používaných komponent

Component Configurator - diagram



Component Configurator



Component Configurator

- **Výhody**
  - Dynamická konfigurace
  - Centrální správa komponent
  - Modularita
- **Nevýhody**
  - Runtime overhead
  - Zvyšuje se komplexita

Component Configurator - použití

- **Použití**
  - Java applets
  - Plug and play
  - Windows Service Control Management(SCM)

Interceptor - Motivace

- **Rozšiřitelnost rozsáhlého systému (frameworku) o nové služby**

- **Řešení:**
  - **Monolitický systém obsahující vše?**
    - Velké, neflexibilní řešení
    - Neznáme všechny potřeby klientů v době vývoje
  - **Úplná otevřenost systému?**
    - Nebezpečné
    - Složité pro uživatele

- **Jiné řešení:**
  - Nadefinovat uvnitř systému „body“, ve kterých uživatel může ovlivnit a rozšířit chování systému.

Interceptor - Motivace

- **Rozšiřitelnost rozsáhlého systému (frameworku) o nové služby**

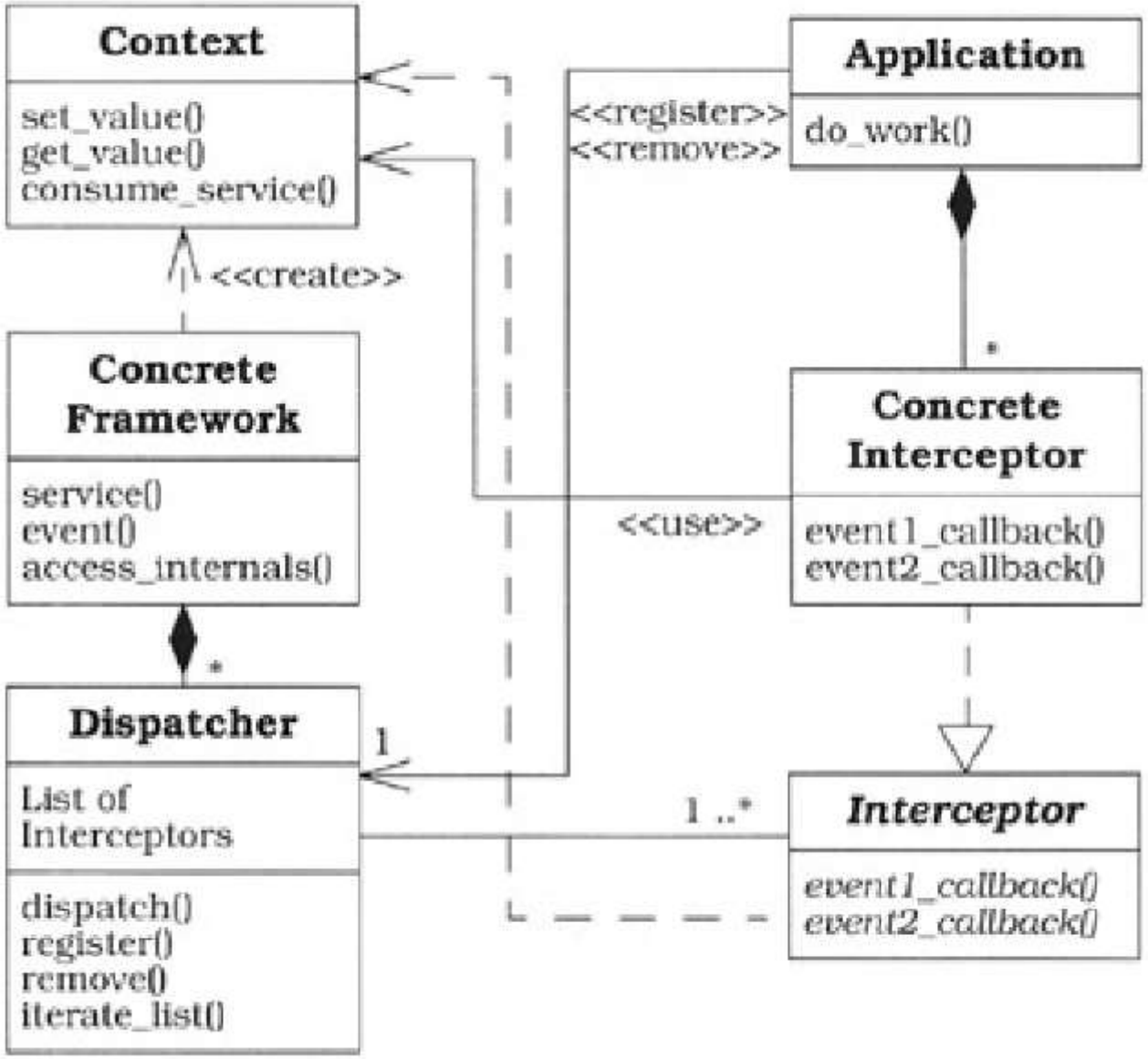
- **Řešení:**
  - **Monolitický systém obsahující vše?**
    - Velké, neflexibilní řešení
    - Neznáme všechny potřeby klientů v době vývoje
  - **Úplná otevřenost systému?**
    - Nebezpečné
    - Složité pro uživatele

- **Jiné řešení:**
  - Nadefinovat uvnitř systému „body“, ve kterých uživatel může ovlivnit a rozšířit chování systému.

Interceptor - Idea

- **Přidávání služeb do systému na přesně určených místech**
  - Místo = událost (příjem zprávy, zpracování dotazu, …)
  - Služba = callback (logování, šifrování, …)
- **Částečné otevření a zpřístupnění vnitřní funkcionality systému**

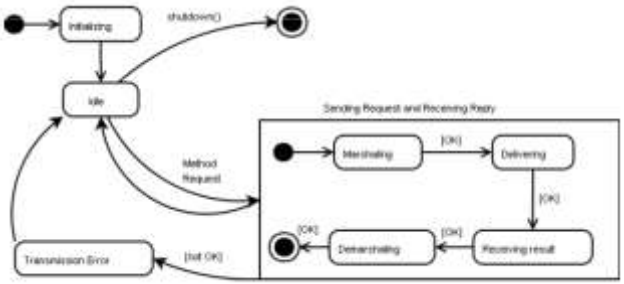
Interceptor – Struktura 2



Interceptor – Struktura 1

- **Concrete framework**
  - Základní služby
  - Události a příslušné dispatchery v klíčových místech
- **Interceptor**
  - Je příslušný události nebo skupině událostí
  - Definuje rozhraní pro integraci vnějších služeb
- **Concrete interceptor**
  - Implementuje rozhraní interceptoru
  - Používá context object pro komunikaci s frameworkem
- **Dispatcher**
  - Umožnuje registraci konkrétních interceptorů
  - Zavolá metody registrovaných interceptorů při události
- **Context object**
  - Pozorování vnitřního stavu frameworku („Accessor“)
  - Ovlivňování chování frameworku („Mutator“)
- **Application**
  - Implementuje konkrétní interceptory a registruje je pomocí dispatcherů

Interceptor – Návrh systému



- **Změna systému na stavový automat**
  - Přechod mezi stavy – potenciální místo pro událost
- **Definování rozhraní pro callback zpracování události**
  - Obecný interceptor
- **Definování rozhraní pro přidávání a odebíraní interceptorů**
  - Dispatcher pro každou událost
- **Vytvoření kontextu pro událost:**
  - Informace o události a vnitřním stavu systému
  - Navíc modifikování chování systému
- **Vztah událost – callback je 1 ku n**
  - Dispatcher rozhoduje o způsobu a pořadí zavolání callbacků.

Interceptor - Výsledek

- **Výhody:**
  - **Rozšiřitelnost a flexibilita**
    - Bez nutnosti znát nebo měnit vnitřní strukturu systému
  - **Přehlednost a jednoduchost pro uživatele**

- **Nevýhody:**
  - **Výzva pro návrháře**
    - Počet událostí a místa pro ně
    - Počet dispatcherů
    - Možnosti kontextů
  - **Chybné interceptory**
  - **Kaskády událostí a zavolání interceptorů**
    - Deadlock

Interceptor – Varianty, příklady použití

- **Varianty**
  - Interceptor proxy aka Delegator
  - Single interceptor per Dispatcher
  - Interceptor factory
    - Místo registrace interceptorů pomocí Dispatcherů konkrétní framework registruje interceptory pomocí klientem dané továrny
  - Implicit Interceptor Registration
    - Například registrace všech Interceptorů, nalezených v .dll v nějaké složce
- **Příklady použití**
  - CORBA
  - COM
  - Pluginy do webových prohlížečů

Interceptor – vztahy k jiným NV

Template method

Lze chápat jako jednoduchou variantu Concrete Frameworku, kde jednotlivé kroky algoritmu jsou Interceptory

Chain of Responsibility

Definuje handlery, které lze chápat jako Interceptory

Na rozdíl od NV Interceptor, zpracování skončí když nějaký handler opravdu zpracuje požadavek

Observer

Lze chápat jako variantu Interceptoru, kde Context object obsahuje jenom informací o události, nikoliv o vnitřním stavu frameworku

Extension interface - Motivace

- **Evoluce rozhraní komponent**
  - V počátcích vývoje je těžké předpovědět, jak a kam se systém rozroste
- **Přidávání funkcionalit – přehuštění rozhraní metodami**
- **Těžké udržovat zpětnou kompatibilitu**
  - Přidávání funkcionalit do rozhraní porušuje funkčnost starých verzí klientského kódu, je potřeba rekompilace
  - Ovšem většinou nové featury opravdu potřebuje jenom několik aplikací

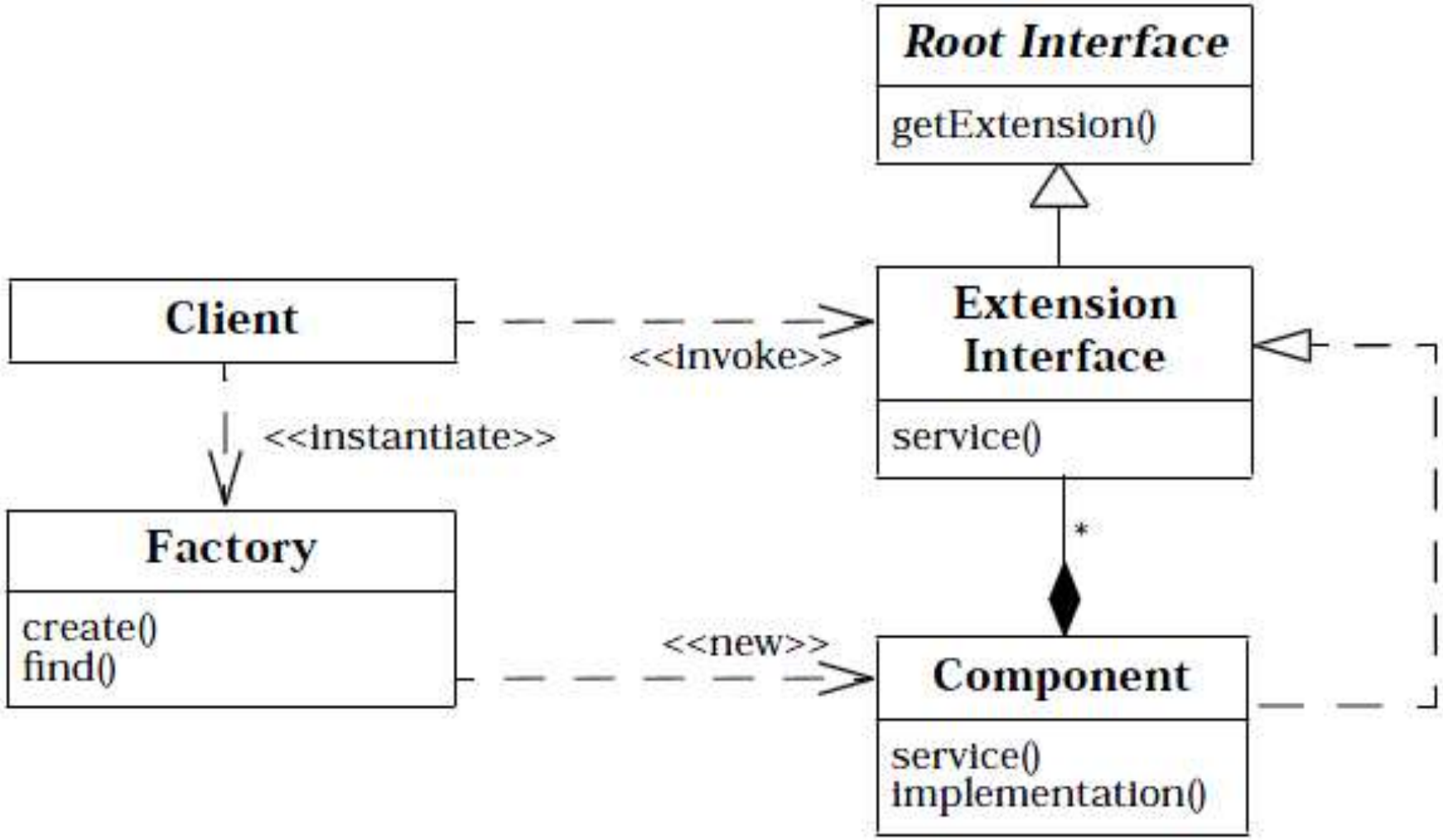
Extension interface - Idea

- **Rozdělit jedno velké rozhraní na vícero malých**
  - Přidávání funkcí = definice dalšího rozhraní
- **Jedna komponenta implementuje několik rozhraní**
  - Update komponenty = implementace dalšího rozhraní
- **Výběr rozhraní pro přístup ke komponentě je na uživateli**
  - Ovšem přístup ke komponentě přímo je zakázán – porušuje zpětnou kompatibilitu
  - Jednotný přístup ke všem rozhraním

Extension interface – Struktura 1

- **Component**
  - Implementuje všechna rozhraní, která podporuje
- **Root interface**
  - Rozhraní, které musí implementovat každá komponenta
  - Přístup k ostatním rozhraním (třeba pomocí identifikátorů rozhraní)
- **Component factory**
  - Definuje funkcionalitu pro vytváření instancí komponent
  - Vrací „root interface“ (klientský kód by neměl pracovat s komponentou přímo)
- **Extension interface**
  - Definuje specifickou funkcionalitu
  - Zároveň hraje roli „root interface“
- **Client**
  - Používá „component factory“ pro vytváření komponent
  - Přístup ke komponentě pouze přes specifická rozhraní

Extension interface – Struktura 2



Extension interface – Implementace

- **Multiple inheritance**
  - Třída komponenty se dědí od všech potřebných extension interface
  - Výhodné pří použiti objektově orientovaných jazyku – downcasting může hrát roli Component Factory
- **Nested classes**
  - Extension interface jsou implementovány uvnitř komponenty jako vnořené třídy (Singleton)
- **Separate interface classes.**
  - Extension interface jsou implementovány jako zvláštní třídy
  - Komunikace s třídou komponenty pomocí NV Bridge nebo Adapter

Extension interface - Výsledky

- **Výhody**
  - Velmi snadná rozšiřitelnost
  - Separation of concern
    - Každé rozhraní hraje specifickou roli
- **Nevýhody**
  - Runtime overhead
  - Složitost návrhu
    - Závisí na použité metodě a jazyce
  - Složitost používání
    - Uživatel musí získat referenci na extension interface před použitím

Extension interface – varianty, příklady

- **Varianty**
  - Extension object
    - Varianta bez použití Component factory pro objektově orientované jazyky
  - Distributed extension interface
    - Vzdálený server obsahuje implementaci komponent
  - Extension interface with access control
    - Klienti mohou mít různá oprávnění na použití rozhraní
  - Asymmetric extension interface
    - Jedno speciální rozhraní pro navigaci mezi rozhraními
- **Příklady**
  - Microsoft COM/COM+
    - Distributed extension interface
  - CORBA component model
    - Asymmetric extension interface

Shrnutí

- **Wrapper Facade**
  - Zapouzdřuje (neobjektové) low-level API
  - Zvyšuje přenositelnost, robustnost
  - Snižuje komplexitu programu
- **Component configurator**
  - Runtimové přilinkování/odlinkování implementací svých komponent
  - Vhodné pro online aplikace, které musí běžet pořád
- **Interceptor**
  - Dynamické přidání funkcionality jako reakci na události
- **Extension Interface**
  - Zabraňuje přehuštění rozhraní velkým množstvím metod
  - Výběr rozhraní pro přístup ke komponentě je na uživateli
  - Jednotný přístup ke všem rozhraním

Otázky