A7B39WPA
Webové podnikové aplikace

Zdeněk Kouba (kouba@fel.cvut.cz)
Petr Křemen (petr.kremen@fel.cvut.cz)

Webové stránky předmětu

- http://cw.felk.cvut.cz/doku.php/courses/a7b39wpa/start

- Podmínky udělení zápočtu
- Pravidla hodnocení práce v semestru
- Vliv hodnocení práce v semestru na výslednou známku
- Samostatná semestrální úloha (externí konzultanti)

- http://cw.felk.cvut.cz/doku.php/courses/a7b39wpa/hodnoceni

Státnicové okruhy

- Modelování a návrh architektury software. Klient-Server. Třívrstvá architektura a její zobecnění na vícevrstvou architekturu. Architektonické vzory, Model-View-Controller (MVC).

- Návrh distribuovaných systémů s použitím webových služeb a servisně orientované architektury (SOA).

Program

1	24.9.	Úvod, klient-server, vícevrstvá architektura
2	1.10.	Protokol HTTP, Servlety
3	8.10.	Architektura a technologie Java EE
4	15.10.	Přístup k databázím, objektově-relační mapování, JPA 2.0
5	22.10.	JPA 2.0
6	29.10.	Spring I
7	5.11.	Spring II, Architektura aplikací, Data access object design
8	12.11.	Přehled hlavních webových frameworků, úvod do Java Server Faces (JSF)
9	19.11.	JSF 2.0
10	26.11.	Pokročilé technologie
11	3.12.	Servisně orientované architektury (SOA), webové služby, distribuované aplikace
12	10.12.	Zabezpečení Java EE aplikací
13.	17.12.	Nasazení enterprise aplikací (deployment, vysoká dostupnost, škálování, monitorování)

Podnikové (enterprise) aplikace

Martin Fowler
Patterns of Enterprise Application Architecture:

"… display, manipulation and storage of large amounts of complex data and the support or automation of business processes with that data."

Podnikové (enterprise) aplikace

- Persistentní data (typicky relační databáze)
- Současný přístup k datům (concurrent access), transakce
- Interaktivní zpracování - rozsáhlé uživatelské rozhraní (mnoho formulářů, webových stránek)
- Dávkové zpracování
- Integrace s jinými podnikovými aplikacemi

Podnikové (enterprise) aplikace

- Integrace s jinými podnikovými aplikacemi
  - Messaging systémy (JMS) – asynchronní zprávy jiným (sub)systémům
  - Remote procedure calls (synchronní zprávy)
    - RPC, RMI
    - CORBA
    - Webové služby
- Integrace s obchodními partnery
  - Transformace dat do jiných syntaktických/ sémantických forem

Podnikové (enterprise) aplikace

- Business “logika” (doménová logika)
  - Business rules
  - Mnoho výjimek

Příklad 1: E-shop

- **Velké množství současně obsluhovaných uživatelů**
  - **škálovatelnost (rozšířením kapacity HW)**
- Doménová logika – jednoduchá
- Tenký klient (web browser)
- Datové zdroje
  - Relační databáze s jednoduchým datovým modelem
  - Komunikace s podnikovým skladovým systémem

Příklad 2: Systém pro zpracování leasingových smluv

- Počet současně pracujících uživatelů poměrně malý
- **Doménová logika složitá**
  - **Předčasné ukončení, pozdní platby, dopravní nehody, pojistné, mnoho produktů, změna podmínek**
  - **Složitý life-cycle => komplikované transakce**
- Datové zdroje
  - Komunikace s obchodními partnery (pojišťovnami, servisy, dodavateli)
  - Databáze se složitým datový model
- User interface
  - Složitý (už proto, že složité transkce)
  - Tenký klient pro zákazníky
  - Rich (tlustý) klient pro úředníky

Příklad 3: Systém pro evidenci nákladů malé firmy

- Malý počet uživatelů
- Jednoduchá doménová logika
- Datové zdroje
  - Databáze s několika málo tabulkami
- User interface
  - Jednoduchá html prezentace

Kde je tedy výzva?

Příklad 3: Systém pro evidenci nákladů malé firmy

- Malý počet uživatelů
- Jednoduchá doménová logika
- Datové zdroje
  - Databáze s několika málo tabulkami
- User interface
  - Jednoduchá html prezentace
- **Rozšiřitelnost**
  - **Výpočet cestovních náhrad**
  - **Reporting finančnímu řediteli**
  - **Rezervace letenek na služební cesty**
  - **Etc.**

Příklady - shrnutí

- E-shop: škálovatelnost
- Správa leasingových smluv – složitá obchodní logika, složitá databáze, udržovatelnost (maintainability)
- Evidence nákladů malé firmy – rozšiřitelnost (přidávání nových funkcionalit)

Výběr architektury:

- Společná architektura?
- Architektonické vzory (patterns)

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
  - Doba, která uplyne od odeslání požadavku do přijetí výsledku zpracování požadavku
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
  - Doba, která uplyne od odeslání požadavku do okamžiku než systém zareaguje na přijetí požadavku (aniž by ho nutně zpracoval)
  - Uživateli často zohledňován více bež doba odezvy
  - Výpisy o průběhu zpracování požadavku
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
  - Minimální čas nutný pro získání odezvy na “NOP” požadavek (nezapočítává se čas vlastního výpočtu)
  - Vyvarovat se RPC na pomalé síti
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
  - Typická jednotka **tps** (počet transkací za sekundu)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
  - typicky kontext pro porovnávání jiných výkonových faktorů
  - Např. doba odezvy při daném počtu současně pracujících uživatelů
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
  - Určuje, jak moc se jiný výkonový faktor mění v závislosti na zátěži
- Kapacita
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
  - Maximální efektivní průchodnost nebo zátěž
- Efektivnost
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
  - Výkon vztažený na počet/velikost zdrojů
  - 30 tps na 2 CPUs je efektivnější než 40 tps na 4 CPUs
- Škálovatelnost

Výkonové faktory

- Doba odezvy (response time)
- Responsiveness
- Latence
- Průchodnost (throughput)
- Zátěž (load)
- Citlivost na zátěž (load sensitivity)
- Kapacita
- Efektivnost
- Škálovatelnost
  - Míra určující,
    - jak přidání zdrojů ovlivní výkon
    - Jak drahé je přidat zdroje
  - Vertikální škálovatelnost (scaling up) – přidání zdrojů jednomu serveru (typicky zvětšení kapacity RAM)
  - Horizontální škálovatelnost (scaling out) – přidání dalších serverů

Výkonové faktory

Ladění výkonových faktorů aplikace versus zvýšení výkonu HW

- Nový (výkonnější) HW je často levnější než ladění aplikace pro dosažení vyššího výkonu na slabším HW
- Přidání dalších serverů (je-li aplikace škálovatelná) je často levnější než zadání úkolu programátorovi

Při vývoji se zaměřit na udržovatelnost (maintainability).

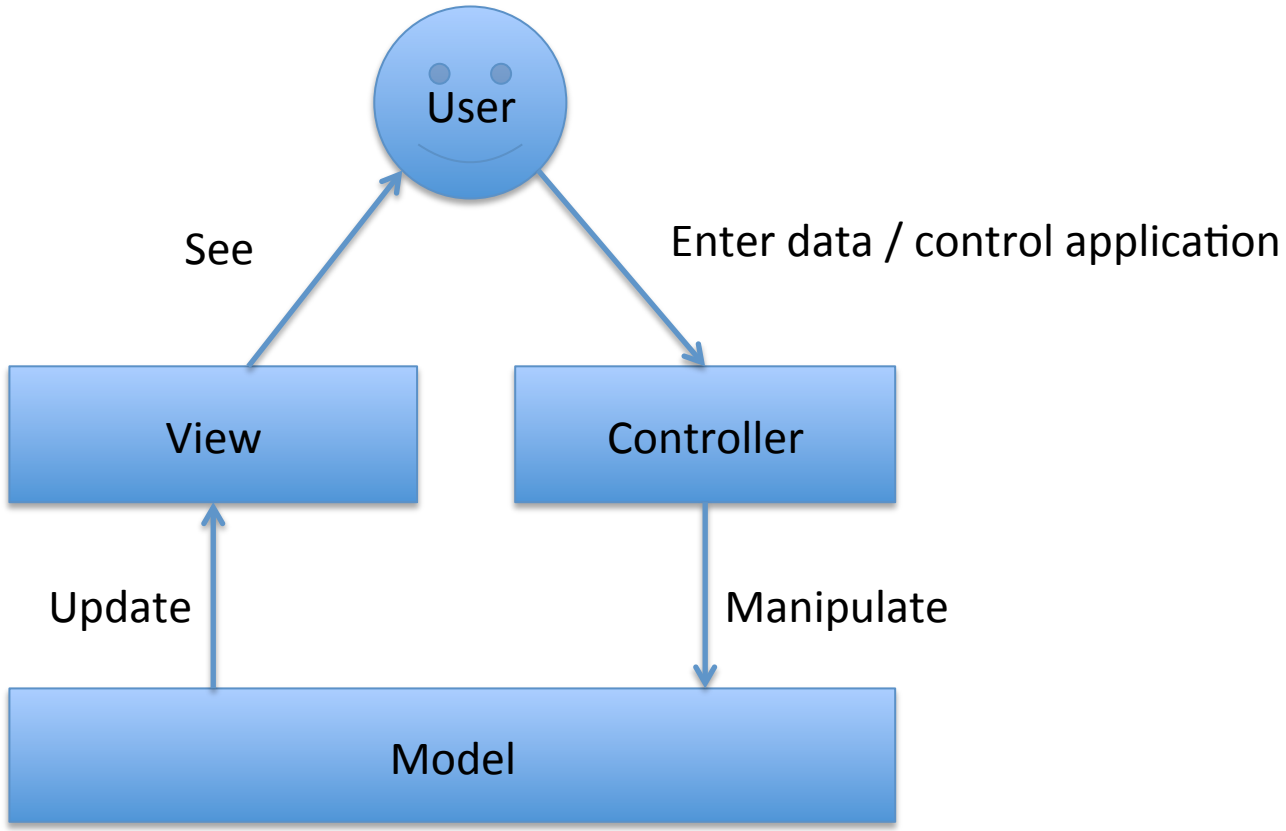
MVC architektura

Model-Controller-View

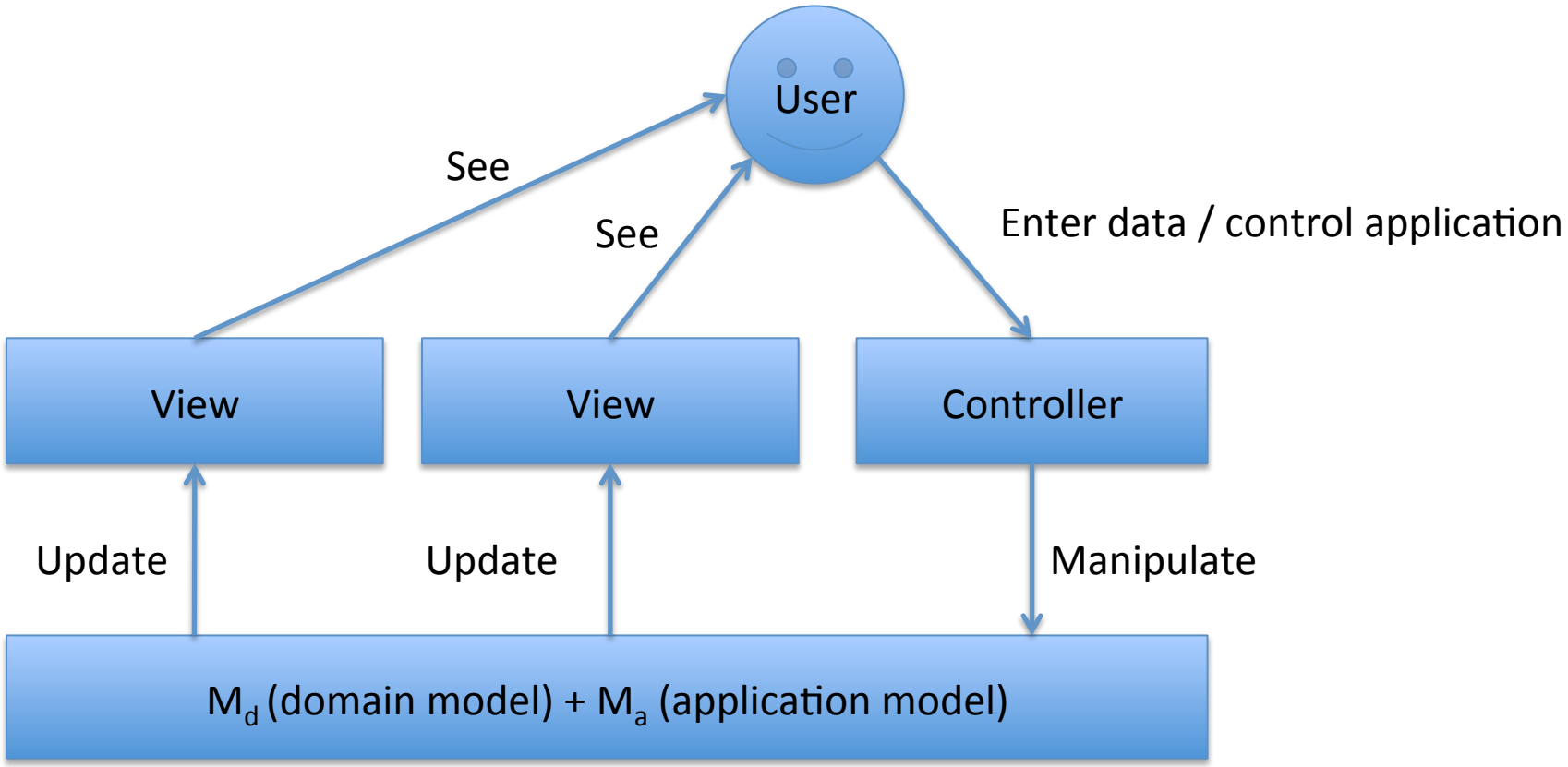
Prof. Trygve Reenskaug – norský prof. computer science – formuloval MVC architekturu v roce 1979 během stáže v Xerox Palo Alto Reseacrch Center (PARC).

Stalo se základním paradigmatem aplikací v jazyce SmallTalk (nejznámější implementace právě od PARC).

MVC architektura



MVC architektura



Vícevrstvá architektura

- Multi-layer nebo multi-tier ?

Vícevrstvá architektura

- Multi-layer nebo multi-tier ?

- Multilayer
  - Logická organizace vrstev – nesouvisí s fyzickým umístěním na tom či onom stroji
- Multi-tier
  - Fyzická organizace vrstev – souvisí s fyzickým umístěním na různých počítačích
  - Zobecnění architektury klient-server
  - 3-tier: tlustý klient (desktop), doménová logika (aplikační server), databáze (databázový server)

3 layer architecture

- Presentation layer
- Domain logic layer
- Data source layer

n layer architecture

- Presentation layer (v případě tenkého klienta: controller + templates)
- Service layer (služby tvořící API k doménové logice)
- Domain logic layer (bussiness objekty)
- Data access layer
- Persistence layer (obvykle relační DB, transparentní)

Striktní/relaxovaná vícevrstvová architektura

- **Striktní**: vyšší vrstva přistupuje pouze k bezprostředně nižší vrstvě
  - Příklad: ISO/OSI síťový model (7 vrstev)

- **Relaxovaná**: vyšší vrstva přistupuje k nižším vrstvám

==========

- Nižší vrstva v **žádném případě** nepřistupuje k vyšším vrstvám (ani je nezná). Pokud ano, jedná se o spaghetti-code.

Business logic invaded into presentation layer

Drawbacks caused by infiltration of parts of business logic into presentation layer:

- difficult or impossible to allow the program to be driven by another program
- difficult automated testing
- difficult transition from interactive to batch mode

Business logic completely isolated from the presentation layer

- automated test scripts can be run against the application (esp. regression testing)
- automated test cases can be created (by business experts) before the GUI finalized
- the application can be deployed in ‘’GUI-less’’ mode, so only the API is available
  - design of complex application suites simplified
  - business-to-business service applications (without human intervention)
- automated function regression tests detect any violation of the promise to keep business logic out of the presentation layer

Domain logic tied to Data Source layer

- database server down (or rework) => the programmers can't work (their work requires the presence of the database)
- database refactoring => domain logic rework

Observation

Domain logic layer shall be isolated both from

- the presentation layer

and

- the data source layer

Hexagonal Architecture

Alister Cockburn [kobérn]

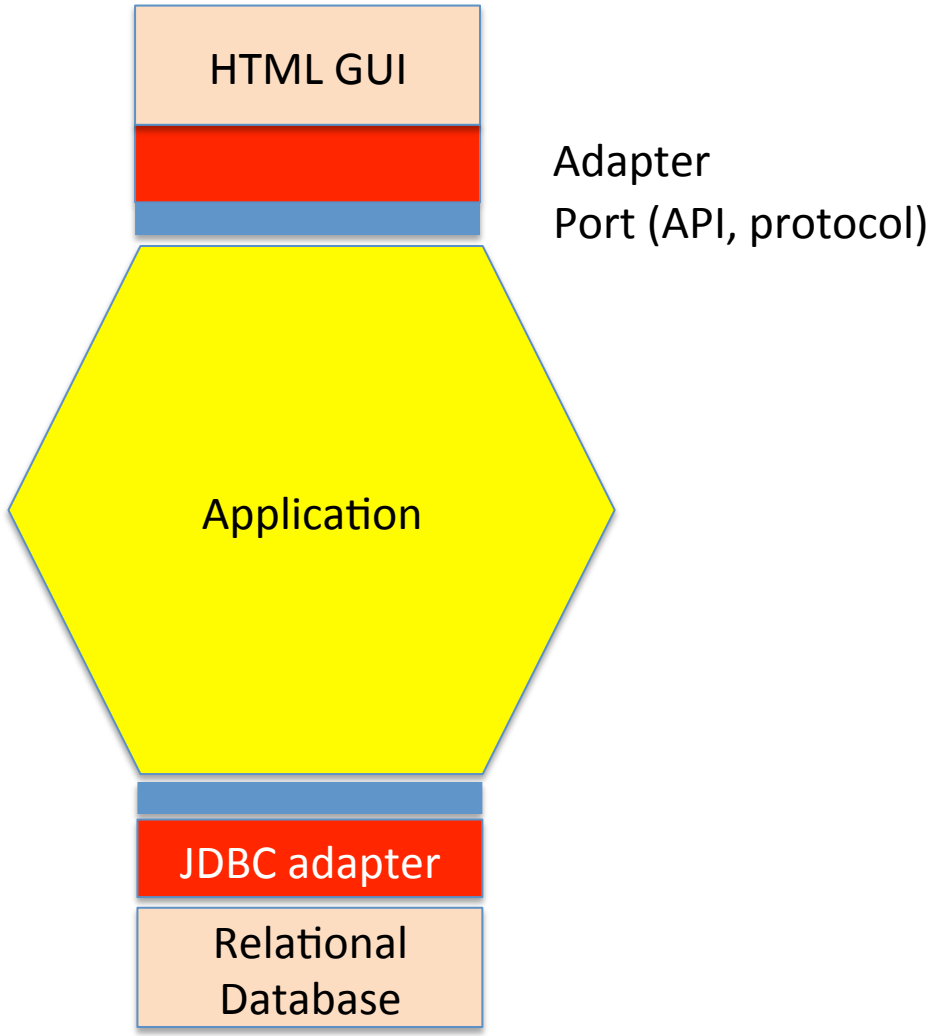
Original article:

http://alistair.cockburn.us/Hexagonal+architecture

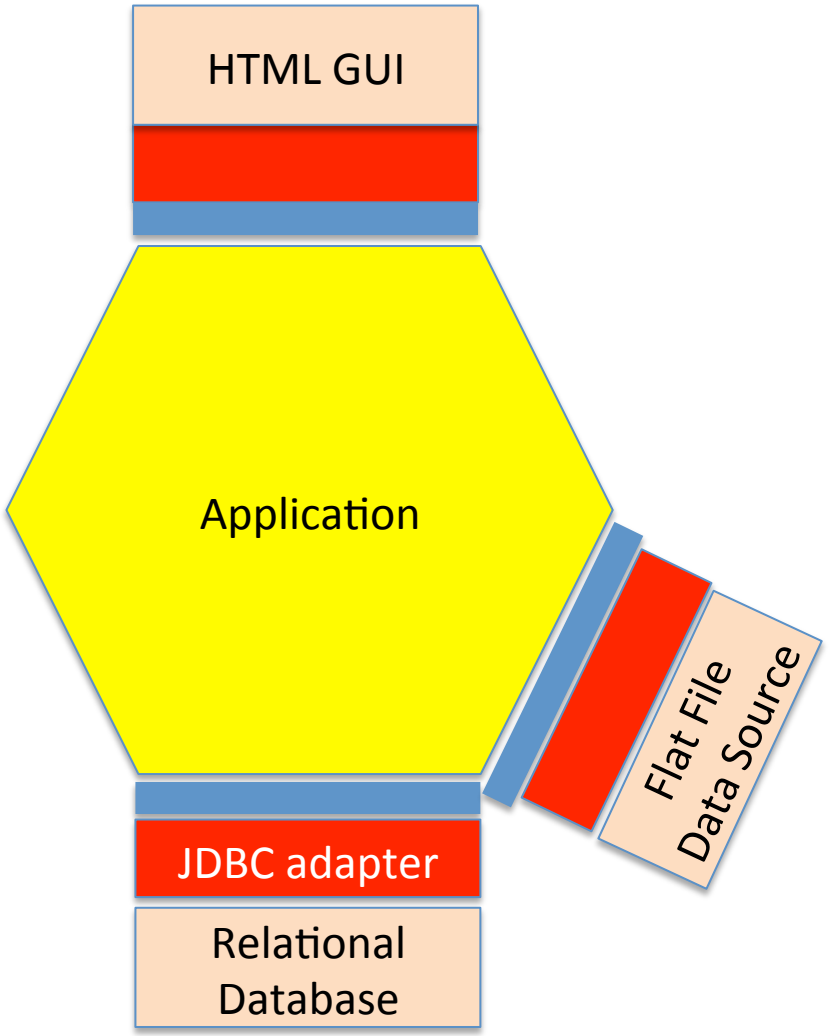
Brief explanation by Andreas Dossier:

http://www.dossier-andreas.net/software_architecture/ports_and_adapters.html

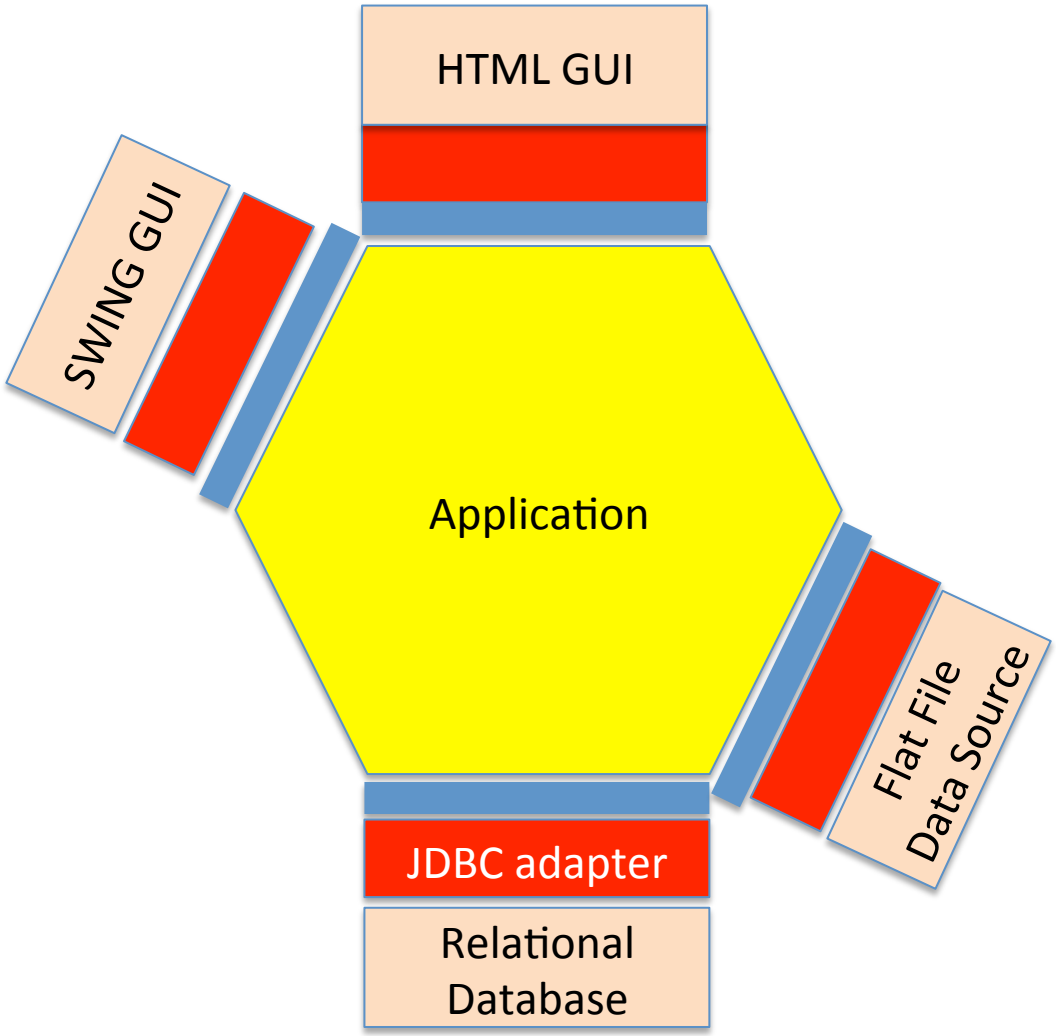
Hexagonal Architecture



Hexagonal Architecture



Hexagonal Architecture



Hexagonal Architecture