Architektura a technologie Java EE, Architektonické vzory, MVC, DAO

- Proč architektura/design?
    - vývoj
    - údržba
    - dokumentace (pro ostatní)
    - efektivita
    - prevence chyb

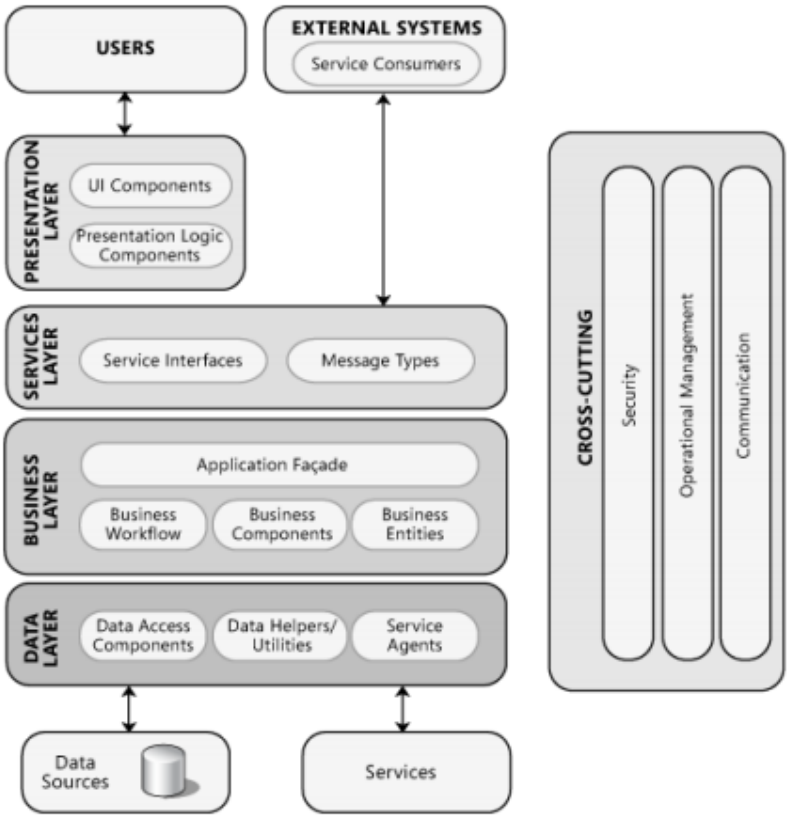
- Proč vzory?
    - best practice
    - lepší struktura
    - komunikace

Software architektura

- popisuje celkovou strukturu systému
- Deployment environment
- Platform and technology specifics
- Expected system scope
- oproti komponent design:
    - mění se málo
    - ovlivňuje celý systém
    - architektonické styly (x DP)

- důležité principy:
    - Separation of concerns
    - Single responsibility principle
    - Law of Demeter (Don’t talk to strangers)
    - Don’t repeat yourself

- většinou vícero architekturních stylů
- měla by jasně vysvětlovat strukturu systému, ale skrývat detaily implementace
    - tzn. kde se dějí věci, ne jak
- měla by se starat jak o funkční, tak nefunkční požadavky



Architektonické styly

- je jich hodně, většinou se kombinují
- mají různé dělení

Communication

Service-Oriented Architecture

- distribuované aplikace poskytující si navzájem služby
- standardní protokoly (REST, HTTP) a formáty dat (JSON, XML)
- Loose coupling (volná spojení), lehká změna implementace
- microservice

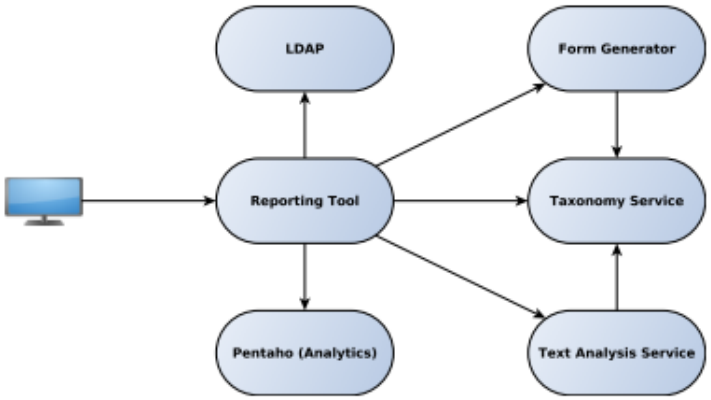
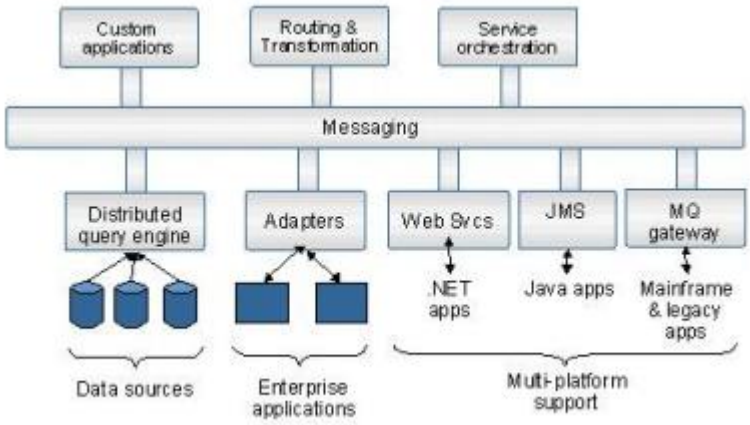


Figure : SOA system example.

Message bus

- centrální fronta zpráv, stará se o jejich distribuci
- asynchronní zprávy mezi klienty
- loose coupling, škálovatelné



Deployment

Klient/server

- klient posílá requesty, server odpovídá
- užití ve web aplikacích

N(3)-tier

- nezávislé vrstvy nabízející funkcionalitu
- lehčeji škálovatelné
- např. company firewall

Domain

Domain-driven Design

- business komponenty reprezentují doménové entity
- vhodné pro modelování komplexních domén
- společný jazyk/model pro vývojáře a domain experts

Structure

Objektově orientovaná

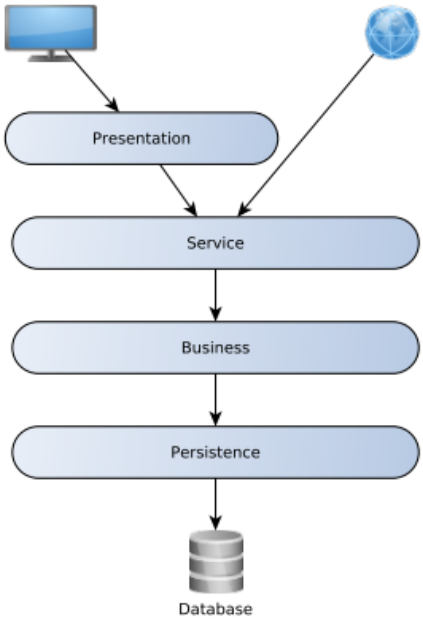
- objekty se skládají jak z chování, tak z dat
- přirozená reprezentace světa
- zapouzdření implementačních detailů

Component-based

- systém dekomponován na logické/funkcionální komponenty
    - nabízí public interface
    - mohou být distribuované
- podporuje SoC, zapouzdření
- vyšší level než OOP

Layered architecture

- vrstvy se související funkcionalitou
- typické pro web aplikace
- zapouzdření chování, čisté SoC, vysoká koheze, loose coupling
- testovatelnost
- na rozdíl od N-tier architektury, vrstvy jsou většinou v jednom procesu (např. AS)
- komponenty komunikují jen s komponentami na stejné či nižší úrovni (strict x loose)
- cross-cutting concerns napříc vrstvami (bezpečnost, logování,…)



Designové vzory

Gang of Four Patterns

vizte OMO

Enterprise Design Patterns

PEAA - Patterns of Enterprise Application Architecture

Data Transfer Object (DTO)

- objekty přenáší data mezi procesy → méně volání

Lazy Load

- objekt hned nezná všechna data, ale ví, jak je načíst → vhodné např. pro velké množství dat

Model View Controller (MVC)

- odděluje UI vykreslování od UI logiky a dat

Unit of Work

- groupování vícero operací do jedné (transakce), ta pak může provést i změny v trvalém úložišti (JPA)

Data Access Object (DAO)

- DA objekt zapouzdřuje veškerý přístup ke zdroji dat
- abstraktní interface skrývá detaily o přístupu k datům (může to být RDBMS,…)

Inversion of Control (IoC)

- nejčastěji u frameworků
- framework přebírá kontrolu nad tím, co a kdy bude instancováno a voláno
- nabídne abstraktní design, my doplníme chování
- zvlášť důležité u aplikací reagujících na nějaké klientské akce
- aka The Hollywood Principle – “Don’t call us. We’ll call you.”

Dependency Injection

- assembler se postará o populating fieldu v nějaké třídě s příslušnou implementací pro cílové rozhraní
- jen řekneme, že chceme závislost určitého typu
- umožňuje aplikaci používat loosely coupled komponenty se zaměnitelnými implementacemi