Logická architektura software

Z FITwiki

Obsah

Softwarová architektura

Softwarová architektura

- je způsob uspořádání SW komponent tak aby, splňovalo různé požadavky (funkční a nefunkční).
- soubor významných rozhodnutí o organizaci SW systémů, výběr strukturních prvků a jejich rozhraní, pomocí nichž je systém sestaven, společně s jejich chováním specifikovaným spoluprácí těchto prvků, seskupení do skupiny větších podsystémů a architektonických styl, který řídí tuto organizaci.

Logická architektura (logical architecture)

- je organizace tříd do balíčků (jmenných prostorů), podsystémů, vrstev. Neříká nic o jejich fyzickém umístění
- v UML modelujeme jako diagram balíčků

Architetura nasazení (deployment architecture)

- opak logické architektury - nasazení subsystémů do různých prostředí, na různé počítače
- v UML modelujeme jako diagram nasazení

Vrstvená architektura

Vrstva

- seskupeni tříd, balíčků, podsystémů, které souvisejí z hlediska zodpovědnosti za nějaký vyznamný aspekt systému.
- vertikální dělení, seskupení z hlediska odpovědnosti, hierarchické

Vrstvená architektura

- organizovaný hierarchicky, vyšší využívá služeb nižší vrstvy
- komplexnost je zapouzdřena ve vrstvách
- lepší podmínky pro vývoj v týmu
- redukce provázanosti, zvýšení koheze
- napomáhá lepší pochopitelnosti systému, přehlednosti a udržovatelnosti
- dovoluje přijmout komunikaci pouze v rámci vrstvy, vynucují viditelné závislosti

Typická 3-vrstvá architektura

- uživatelské rozhraní
- aplikační logika a objekty domény
- technické služby (databáze, záznam chyb,…)

Sekce
: horizontální dělení, dělení na relativně paralelní podsystémy

Striktní (uzavřená) vrstvená architektura
: vyšší vrstva může používat jen bezprostředně nižší vrstvu

Uvolněná (otevřená) vrstvená architektura
: vyšší vrstva používá několik nižších vrstev

Doménový objekt
: třídy odpovídající reálné doméně, mají přiřazenu část zodpovědnosti aplikační logiky

Závislosti vrstev a balíčků

Balíček
: (dle UML) je seskupení elementů modelu pod jedním jménem. Může obsahovat jiné balíčky a entity.

Závislost
: Balíček A závisí na balíčku B jestliže změny v balíčku B mohou vynutit změny v balíčku A

- závislost metod -> závislost tříd -> závislost balíčků -> závislost vrstev

Závislost metod
: vzniká voláním v kódu.

Závislost tříd
: je hlubšího rázu, třída většinou závisí na třídě v jiném balíčku nebo vrstvě a tím zesložiťuje závislosti balíčků a vrstev.

Závislost vrstev
: by měla být směrem od vyšší k nižší. Nižší vrstvy by měly být stabilní (neměnit se příliš v čase).

Zdroje závislostí

- Dědění
- Asociace
- Volaní operace
- Parametr operace
- ...

Cyklická závislost
: A závisí na B, které závisí na A (i tranzitivně přes další třídy)

Odstranění cyklické závislosti

- **mezi balíčky** - A a B se provádí přidáním speciálního balíčku A2, který obsahuje prvky A takové, že je na nich B závislé. Na novém balíčku A2 je pak závislé A i B. Odstranil se tak jeden směr závislosti, druhý zůstává nezměněn.
- **mezi třídami** - pracuje na principu vytvoření nového rozhraní, na které se závislost předá (tzv. Presenter). Je také možné vytvořit rozhraní pro snížení počtu závislostí.

Architektonické vzory

Princip oddělení pohledu a modelu

Vrstvy uživtelské rozhraní se často mění (různé pohledy na data, různé požadavky uživatelů, ...)

Podstata:

- vztah vrstvy UI (pohledu) k ostatním vrstvám:
  - Nevytvářet vazbu objektů nižších vrstev na objekty uživatelského rozhraní
  - Nevkládat aplikační logiku do operací tříd UI
  - Objekty nižších vrstev by neměli přímo znát objekty UI a posílat jim zprávy.
- Řešení
  - pomocí asynchronních událostí:
  - Použití návrhového vzoru Observer.

Model-View-Controller (MVC)

je architektonický vzor, rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní.

- **Model** - reprezentuje data, odpovídá vrstvě domény, obsahuje objekty domény, přidává k datům aplikační logiku
- **View** - reprezentuje způsob zobrazení dat, zodpovědná za prezentaci modelu (objekt UI, dynamické HTML stránky)
- **Controller** - reprezentuje komunikaci v systému, zpracovává reaguje na události (fyzicky uživatele)

Prima zavislost view->model, controler->view, controler->model. Nepriama zavislost (observer) model->view

PCMEF

- **Presentation** - uživatelské rozhraní
- **Controller** - zpracování požadavků
- **Mediator** - práce s daty
- **Entity** - ukládání aktuálně zpracovávaných dat
- **Foundation** - přímá komunikace s databází

Servisně orientovaná architektura (SOA)

Service-oriented Architecture (SOA)
: je jistým stylem vytváření informačních systémů, který reflektuje business proces. Model je založen na komunikaci mezi poskytovatelem služeb a spotřebitelem služeb

- volné propojení
- nezávislost služeb,
- abstrakce
- znovupoužitelnost
- **bezestavovost**
- nezávislost na platformě (multiplatformní)

SoA vs. Klient-Server
: nesouvisejí, SoA je plně distribuovaná, spotřebitel je také služba (druhé strany), K-S jsou jednostranné a propojené dvě entity.

Role komunikujících stran

- **poskytovatel služeb (service provider)** implementuje a nabízí služby, služba je specifikovaná svým popisem
- **spotřebitel služeb (service consumer)** na základě popisu vyhledá službu v registru služeb a použije ji
- **registr služeb (service register)** podle něj si spotřebitel služby vyhledá

Spolupráce mezi službami

- služby nekomunikují pouze se spotřebitelem, ale i mezi sebou:
- služby poskytují své prostředky bud' přímo cílovému spotřebiteli anebo jiným službám,
- služby mezi sebou spolupracují (komunikují) zasíláním zpráv.

Typy spolupráce mezi službami

- **Kooperace** – služba využívá prostředky jiné služby pro realizaci nabízených funkcí (sama služba něco dělá ale nedělá vše sama)
- **Agregace** – služba tvořená pouze jinými službami (spojení), sjednocení výsledků, nová služba sestavená ze dvou (nebo více) služeb nabízí kombinaci funkcí dílčích služeb (sama služba nic nedělá jen spojuje)
- **Choreografie** – spolupráce služeb napříč organizacemi, služby spolupracují za účelem provedení business procesu. (série volání několika služeb)
- **Orchestrace** – služba řídí součinnost ostatních služeb za účelem provedení své části business procesu.

Vrstvy SOA

Struktura SoA je mnohovrstevnatá

- **Vrstva business procesů** – BP je posloupnost kroků reprezentující business pravidla a vedoucí k zisku (hmotnému i nehmotnému), reprezentován sekvencí provedení několika služeb (choreografie služeb)
- **Vrstva služeb** – rozhraní jednotlivých komponent sjednocena do služeb, služba za běhu sestavuje komponenty a přeposílá jim požadavky, služba na rozhraní zpřístupňuje své funkce (popis služby)
- **Vrstva komponent** – základní stavební kameny služeb, realizace funkčnosti služeb a zajištění požadované kvality služeb (QoS), komponenty jsou černé skříňky a jejich funkce jsou přístupné pouze rozhraní

Servisně orientovaná analýza a návrh (SOAD)

postup návrhu systému, kombinuje OO, rámce pro podnikovou architekturu a business process modelling

- Identifikace služeb, Klasifikace služeb, Analýza podsystémů, Specifikace komponent, Sestavení služeb, Realizace služeb

Web Services (WS)

technologie pro implementací SOA (spravuje W3C)

Postaveny na technologiích:

- SOAP (simple object access protokol/ servise oriented access protocol) - komunikace spotřebitel-poskytovatel
- Universal Description, Discovery and Integration (UDDI) - registrace a vyhledávání služeb, adresář služeb
- Web Services Description Language (WSDL) - XML popis webových služeb (jaké funkce služba poskytuje, kde je umístěna, jak komunikuje)

SOAP

základní vrstva WS technologie

- výměna XML zpráv mezi spotřebitelem a poskytovatelem přes HTTP
- bezstavový protokol
- podporuje několik typů volání funkcí (notification, one way, request-response) služeb

- definuje strukturu zprávy

Kategorie: Státnice MIS | Analýza a návrh informačních systémů | Státnice AIS | Státnice 2011

- Stránka byla naposledy editována 10. 6. 2013 v 13:29.