Architectural design patterns - MVC, Multitier, Monolithic, P2P, Client/Server

*předmluva: Snadné téma, ale pozor na čas. Nikam nespěchejte, na dané téma se bohužel nedá mluvit do nekonečna. Pokud zvládnete o každém z tématů mluvit lehce pod tři minuty, je to v suchu.*

*Andreas Dumalas*

## Co je to design pattern?

Design pattern jakožto architektura je způsob návrhu řešení problému vynalezený architektem Christopherem Alexandrem. Původně zaměřené na pole jeho působnosti – architekturu, brzy si našlo cestu do mnoha dalších oborů.

## MVC

MVC je zkratka pro Model-View-Controller. Jeho hlavní myšlenka je, že rozdělíme logiku aplikace do tří částí. Tyto tři části mají na starosti uživatelské rozhraní, řídící logiku a datový model. Jeho hlavním účelem je zajistit dlouhodobou udržitelnost kodu.

Díky těmto vlastnostem se stal oblíbeným design patternem nejen pro desktopové využití, ale stal se i oblíbeným návrhovým vzorem webařů.

Jak jsem již zmínil, MVC se skládá ze tří částí:

* Model
* View
* Controller

Tyto tři části dohromady tvoří jeden celek, kde Model zajišťuje práci s daty, většinou komunikaci s databází. View obstarává pohled uživatele na data. Poslední zmíněnou částí je Controller, jehož role je reagovat na změny dat a události vyvolané uživatelem. Na jeho základě je aktualizován View.



## Monolithic Architecture

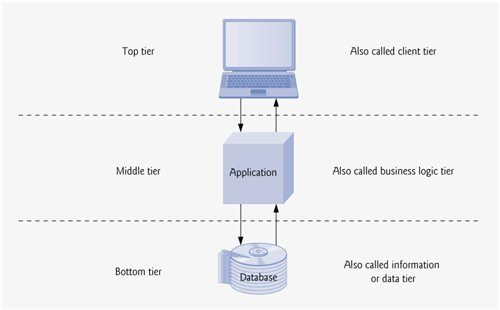
Monolitická architektura je typická pro centralizované zpracování. V monolitické architektuře jsou tři skupiny funkcí v programu vzájemně propleteny a samotný program běží na jednom počítači. Výhodou tohoto řešení je snadné zajišťování ochrany funkcí a dat aplikace před neautorizovaným použitím a snadné zajišťování aplikace proti výpadkům. Monolitická architektura, ale s sebou přináší i řadu problémů:

* obtížná údržba. Tím, že jednotlivé skupiny funkcí nejsou v aplikaci separovány, znamená například přechod na jinou formu komunikace nebo přechod na jiný systém řízení báze dat zásah do mnoha míst programu, což zvyšuje jak časové nároky na údržbu, tak riziko chybných nebo neúplných oprav,
* obtížná přenositelnost aplikace mezi různými platformami. Velmi často jsou aplikace s monolitickou architekturou spojeny s jednou platformou jako pupeční šňůrou. Když se na trhu objeví levnější a výkonnější hardware a/nebo základní software, je velmi nesnadné přenést aplikaci do tohoto nového prostředí.

Monolitická aplikace je v podstatě obrovský, masivní kus kódu plný závislostí. Problém je, že vám při každém vydání nové verze bude klást překážky.

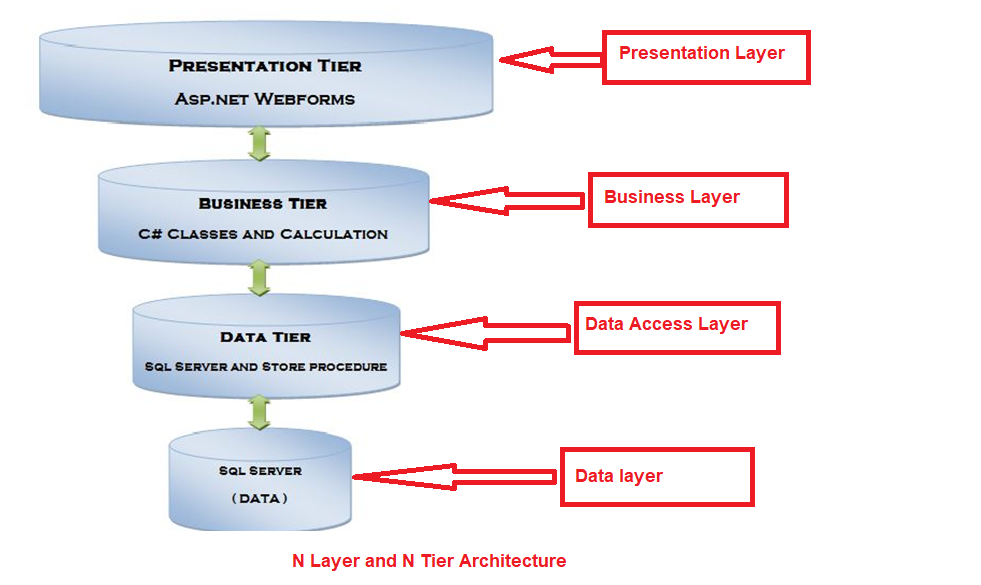
## Multi-tier architecture

Multi-tier, česky označován jako vícevrstvá architektura zajišťuje svoji funkčnost tak, že aplikace je rozdělena do několika vzájemně komunikujících vrstev, jenž spolupracují na definovaném rozhraní. Nejběžnějším příkladem bývá architektura třívrstvá. V takovém případě se jedna vrstva stará o uživatelské rozhraní, druhá o vlastní logiku a třetí o databázi.



Za předpokladu, že aplikaci budeme od začátku vyvíjet jako vícevrstvou, můžeme se kdykoli v budoucnu rozhodnout umístit jednotlivé logické vrstvy (někdy anglicky layers) na určité fyzické stroje (anglicky tiers). Tomu se říká vertikální přístup a určité fyzické stroje jsou předem vyhrazeny k provádění specifických úloh. Je možné také aplikovat horizontální přístup, který v podstatě spočívá v tom, že rozšíříme specifickou fyzickou (hardware) vrstvu.

Je možné také aplikovat kombinovaný postup, který je přiléhavě nazývaný diagonální. Tím rozšiřujeme oba typy vrstev (tier i layers)( hardware i software) dle potřeby.



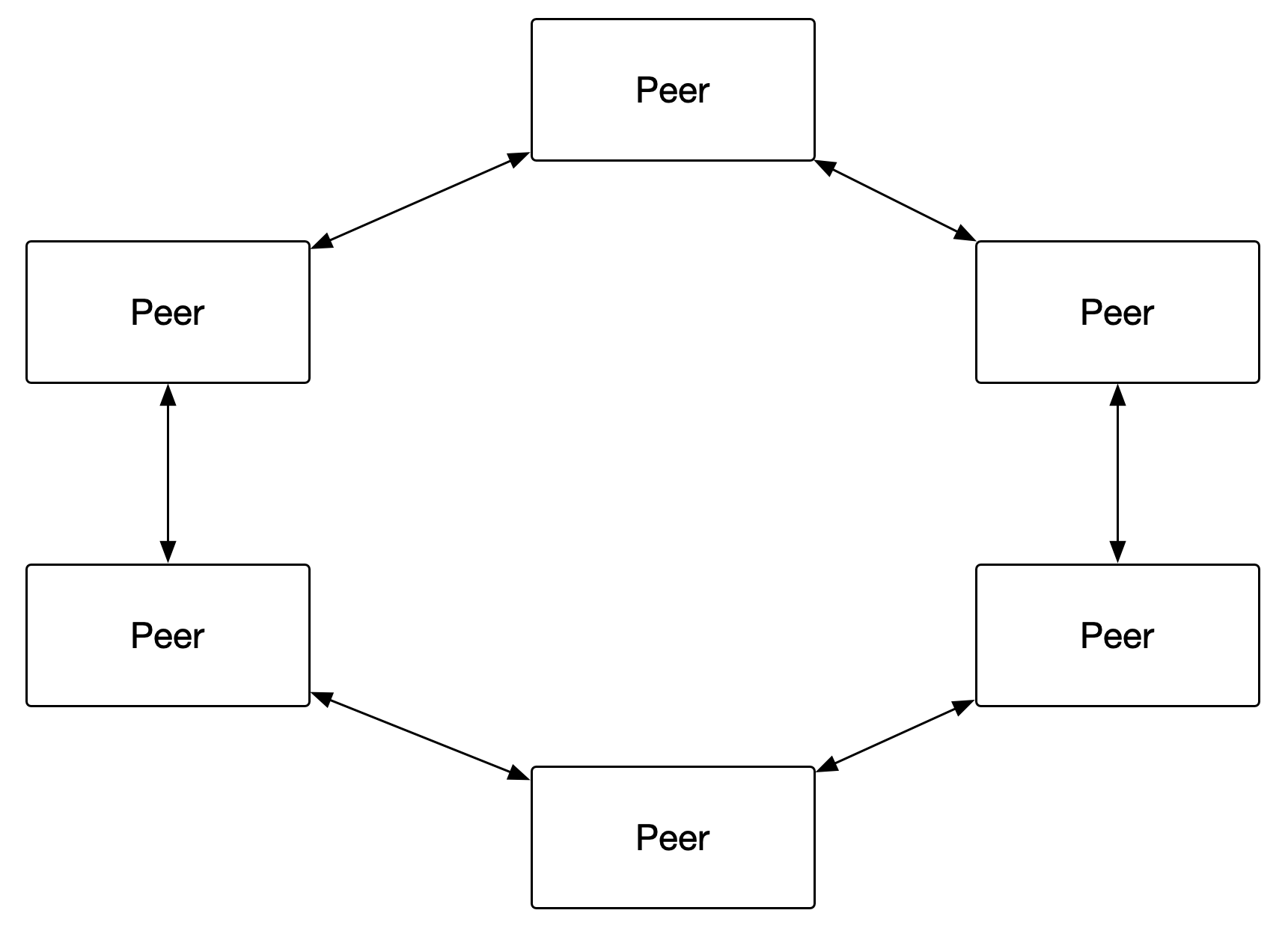
## P2P - Peer to peer

Peer to peer je architektura, která se skládá ze skupiny uzlů, které všechny mají stejnou roli, vykonávají stejnou funkci. To je hlavní rozdíl například proti architektuře Client-server. Pro tento návrhový vzor neexistuje žádný centrální ovládací prvek, všechny uzly jsou si rovny. Všechny uzly pracují zároveň jako přijímač a vysílač dat a zdrojů.

Tento vzor má řadu výhod, některé z nich si vyjmenujeme:

* jsou zapotřebí jen dva uzly
* další uzly se mohou kdykoli připojit
* uzly se mohou kdykoli odpojit

Architektura je ale hodně zranitelná, neboť stačí narušit jeden uzel a tím položíme celou strukturu. Toho využívají především DoS útočníci (Denial of Service). Je složitější administrace této decentralizované sítě. Je jednodušší po síti šířit virusy jako jsou Trojský kůň, spyware či už zmíněného DDOS . Je zde složité provádět obnovu dat či backup.



## Client/Server

Klient server je architektura operující na principu vztahu mezi dvěma programy, v nichž jeden je klient, jenž žádá o služby, a druhý je server, jenž služby poskytuje. Na tomto principu běží například email, některé videohry (jako server pak slouží hráč, který hru hostuje), web, přístup k databázi a další. Dobrým příkladem je webový prohlížeč, kdy klientský program na počítači uživatele může přistupovat k informacím na libovolném webovém serveru na světě. Pokud byste chtěli zkontrolovat zůstatek na svém bankovním účtu, váš prohlížeč předá váš dotaz webovému serveru vaší banky, ten předá dotaz databázovému programu, který pošle dotaz databázovému serveru. Odsud je hodnota zůstatku na vašem bankovním účtu vrácena zpět databázovému systému, který jej pošle do vašeho prohlížeče a výsledný zůstatek zobrazí.

Model klient/server se stal jedním z nejpoužívanějších návrhových vzorů. Používá ho většina obchodních a firemních aplikací. Je také užíván například pro internetové protokoly HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), Telnet (teletype network), DNS (Domain Name Server) a další. Každý klient může poslat žádost o data jednomu, či více připojeným serverům. Na druhé straně, servery mohou tuto žádost přijmout a klientovi vrátit požadovanou informaci. Princip Client/Server může být použitý různými způsoby, základ ale zůstává stejný.

>>Otázku můžete rozšířit o vědomosti nasbírané v otázce 21. Síťová komunikace<<

Výhody: je jednoduché tuto architekturu spravovat narozdíl od p2p, server smí omezit určitým uživatelům přistup k určitým zdrojům či naopak. Lépe zde dokážeme zabezpečit přístup k datům tím pádem hůře by se skrze server rozmnožoval nevyžádaný malware.

Nevýhody: Klient může čekat déle na odpověď, pokud je právě server více vytížený. Vyžaduje více zdrojů a výkonu, protože server musí odpovídat na všechny dotazy od klientů.

