



# Kapitola 6 – Návrh architektúry

# Architektonický dizajn



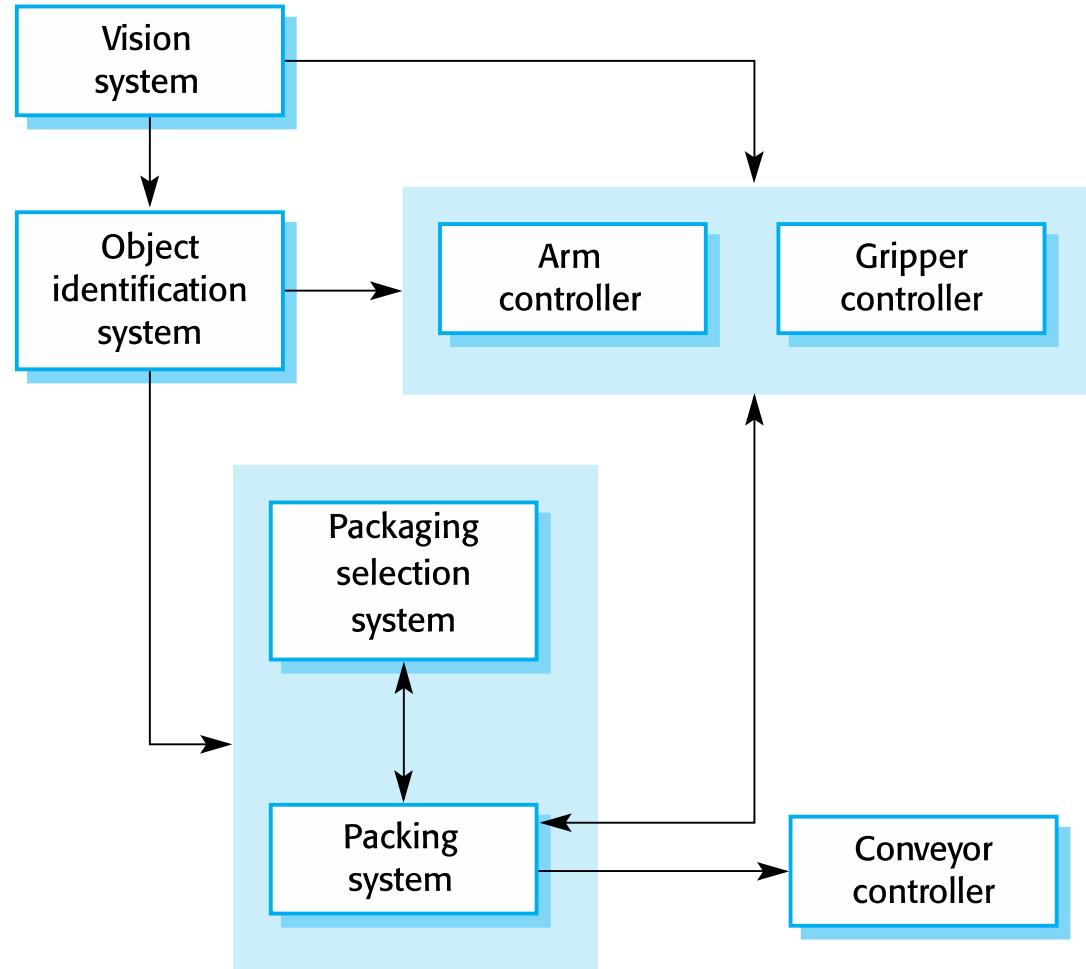
- ✧ Architektonický dizajn (návrh) sa zaoberá pochopením toho, ako by mal byť softvérový systém organizovaný, a navrhnutím celkovej štruktúry tohto systému.
- ✧ Architektonický dizajn je kritickým spojením medzi návrhom a inžinierstvom požiadaviek, pretože identifikuje hlavné konštrukčné komponenty v systéme a vzťahy medzi nimi.
- ✧ Výstupom procesu architektonického návrhu je **model architektúry**, ktorý popisuje, ako je systém organizovaný ako súbor komunikujúcich komponentov.

# Agilný prístup a architektúra



- ✧ Všeobecne platí, že počiatočným štádiom agilných procesov je návrh celkovej architektúry systému.
- ✧ Refaktorovanie architektúry systému je zvyčajne drahé, pretože ovplyvňuje veľa komponentov v systéme.

# Architektúra riadiaceho systému baliaceho robota



# Architektonická abstrakcia



- ✧ Architektúra v malom sa zaoberá architektúrou jednotlivých programov. Na tejto úrovni sa zaoberáme spôsobom, akým je individuálny program rozložený na komponenty.
- ✧ Architektúra vo veľkom sa zaoberá architektúrou komplexných podnikových systémov, ktoré zahŕňajú iné systémy, programy a programové komponenty. Tieto podnikové systémy sú distribuované na rôznych počítačoch, ktoré môžu vlastniť a spravovať rôzne spoločnosti.

# Výhody explicitnej architektúry



## ✧ Komunikácia so zainteresovanými stranami

- Architektúra môže byť použitá pre diskusiu medzi zainteresovanými stranami systému.

## ✧ Systémová analýza

- Znamená to, že je možná analýza, či systém dokáže splniť svoje nefunkcionálne požiadavky.

## ✧ Opäťovné použitie vo veľkom meradle

- Architektúra môže byť opakovane použiteľná v celom rade systémov
- Môžu byť vyvinuté architektúry produktového radu.

# Použitie architektonických modelov



- ✧ Ako spôsob uľahčenia diskusie o návrhu systému
  - Architektonický pohľad na systém na vysokej úrovni je užitočný pre komunikáciu so zainteresovanými stranami systému a plánovanie projektu, pretože nie je preplnený detailmi. Zainteresované strany sa s tým môžu stotožniť a pochopiť abstraktný pohľad na systém. Potom môžu diskutovať o systéme ako celku bez toho, aby boli zmätení detailmi.
- ✧ Ako spôsob dokumentovania architektúry, ktorá bola navrhnutá
  - Cieľom je vytvoriť kompletný model systému, ktorý zobrazuje rôzne komponenty v systéme, ich rozhrania a prepojenia.

# Opäťovné použitie architektúry



- ✧ Systémy v rovnej doméne majú často podobné architektúry, ktoré odrážajú koncepty domény.
- ✧ **Aplikačné produktové rady** sú postavené na základnej architektúre s variantmi, ktoré uspokoja konkrétné požiadavky zákazníkov.
- ✧ Architektúra systému môže byť navrhnutá podľa jedného alebo viacerých architektonických vzorov alebo „štýlov“.
  - Tie zachytávajú podstatu architektúry a môžu byť vytvorené rôznymi spôsobmi.

# Architektonické reprezentácie



- ✧ Krabicové a čiarové diagramy sú veľmi abstraktné - nezobrazujú povahu vzťahov komponentov ani externe viditeľné vlastnosti podsystémov.
  - Užitočné však pre komunikáciu so zainteresovanými stranami a pre plánovanie projektov.
- ✧ Niektorí ľudia tvrdia, že Unified Modeling Language (UML) je vhodnou notáciou na popis a dokumentáciu systémových architektúr.
- ✧ Architektonické popisné jazyky (ADL) boli vyvinuté, ale nie sú široko používané

# Rozhodnutia o architektonickom návrhu



- ✧ Architektonický návrh je kreatívny proces, takže proces sa líši v závislosti od typu vyvíjaného systému.
- ✧ Avšak množstvo spoločných rozhodnutí pokrýva všetky procesy návrhu a tieto rozhodnutia ovplyvňujú **nefunkcionálne** charakteristiky systému.

# Rozhodnutia o architektonickom dizajne



Is there a generic application architecture that can act as a template for the system that is being designed?

How will the system be distributed across hardware cores or processors?

What architectural patterns or styles might be used?

What will be the fundamental approach used to structure the system?

What strategy will be used to control the operation of the components in the system?

How will the structural components in the system be decomposed into sub-components?

What architectural organization is best for delivering the non-functional requirements of the system?

How should the architecture of the system be documented?

# Charakteristiky architektúry a systému



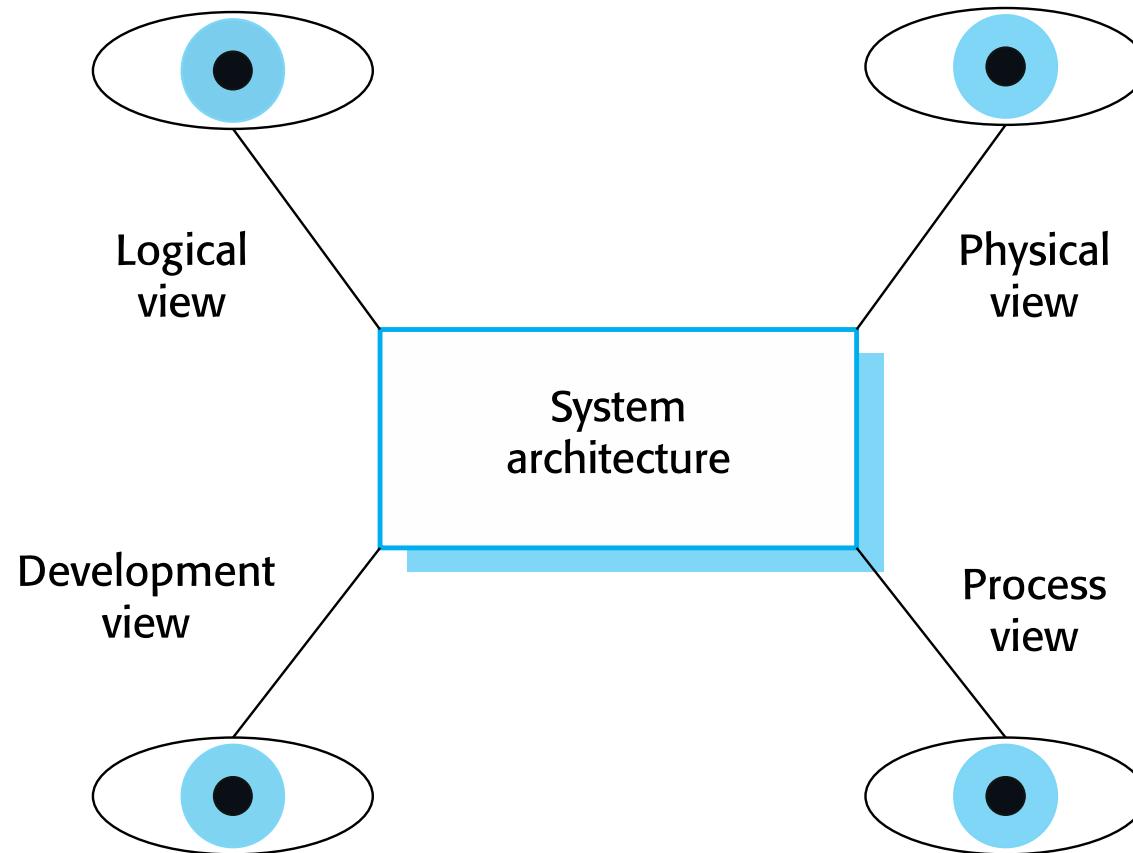
- ✧ Výkon
  - Lokalizovať kritické operácie a minimalizovať komunikáciu. Používajte skôr veľké ako malé komponenty.
- ✧ Bezpečnosť
  - Použite vrstvenú architektúru s kritickými aktívami vo vnútorných vrstvách.
- ✧ Ochrana
  - Lokalizácia prvkov dôležitých pre ochranu v malom počte podsystémov.
- ✧ Dostupnosť a odolnosť
  - Zahrňte redundantné komponenty a mechanizmy na odolnosť voči chybám.
- ✧ Udržiavateľnosť
  - Používajte malé vymeniteľné komponenty.



# Architektonické pohľady

- ✧ Aké pohľady alebo perspektívy sú užitočné pri navrhovaní a dokumentovaní architektúry systému?
- ✧ Aké označenia by sa mali použiť na opis architektonických modelov?
- ✧ Každý architektonický model zobrazuje iba jeden pohľad alebo perspektívu systému.
  - Môže to ukázať, ako je systém rozložený na moduly, ako sa vzájomne ovplyvňujú run-time procesy alebo rôzne spôsoby, akými sú systémové komponenty distribuované v sieti. Pre návrh aj dokumentáciu zvyčajne potrebujete prezentovať viacero pohľadov na architektúru softvéru.

# Architektonické pohľady



# 4 + 1 pohľadový model softvérovej architektúry



- ✧ **Logický pohľad**, ktorý zobrazuje kľúčové abstrakcie v systéme ako objekty alebo triedy objektov.
- ✧ **Procesný pohľad**, ktorý ukazuje, ako sa systém za behu skladá z interagujúcich procesov.
- ✧ **Vývojový pohľad**, ktorý ukazuje, ako je softvér rozložený počas vývoja.
- ✧ **Fyzický pohľad**, ktorý zobrazuje hardvér systému a spôsob, akým sú softvérové komponenty distribuované medzi procesormi v systéme.
- ✧ **Súvisiace prípady použitia alebo scenáre (+1)**

# Architektonické vzory



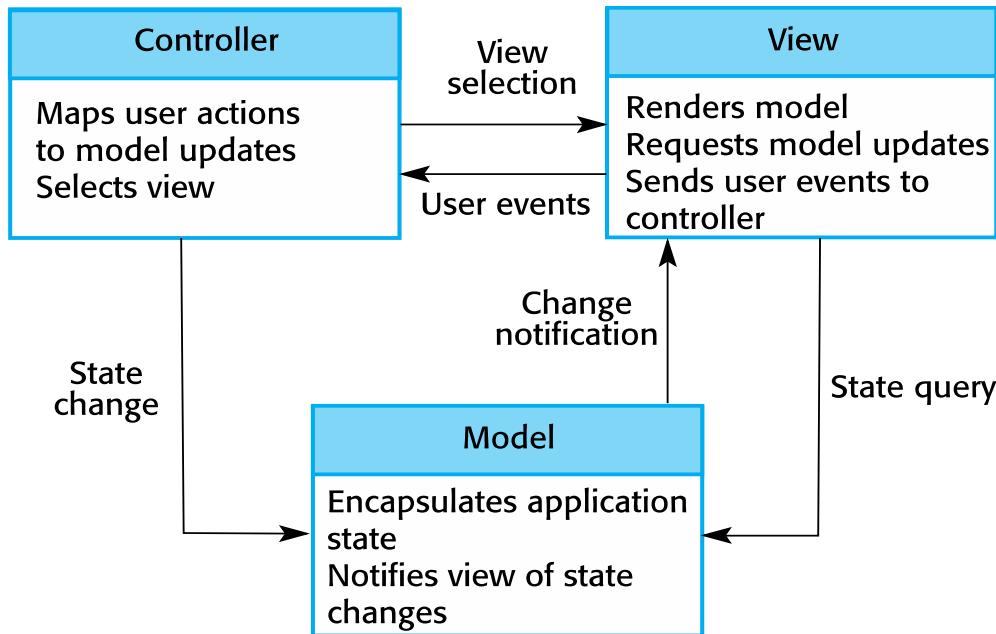
- ✧ Vzory sú prostriedkom na reprezentáciu, zdieľanie a opäťovné použitie vedomostí.
- ✧ **Architektonický vzor** je štylizovaný popis dobrej dizajnérskej praxe, ktorý bol vyskúšaný a testovaný v rôznych prostrediach.
- ✧ Vzory by mali obsahovať informácie o tom, kedy sú a kedy nie sú užitočné.
- ✧ Vzory môžu byť znázornené pomocou tabuľkových a grafických popisov.

# Vzor Model-View-Controller (MVC)



názov	MVC (Model-View-Controller )
Popis	Oddeľuje prezentáciu a interakciu od systémových údajov. Systém je štruktúrovaný do troch logických komponentov, ktoré sa navzájom ovplyvňujú. Komponent Model spravuje systémové údaje a súvisiace operácie s týmito údajmi. Komponent View definuje a riadi spôsob, akým sú údaje prezentované používateľovi. Komponent Controller riadi interakciu používateľa (napr. stlačenie klávesov, kliknutie myšou atď.) a tieto interakcie odovzdá do zobrazenia a modelu. Pozri obrázok 6.3.
Príklad	Pozrite si knihu pre webový aplikačný systém organizovaný pomocou vzoru MVC.
Pri použití	Používa sa, keď existuje viacero spôsobov zobrazenia údajov a interakcie s nimi. Používa sa aj vtedy, keď nie sú známe budúce požiadavky na interakciu a prezentáciu údajov.
Výhody	Umožňuje dátam meniť sa nezávisle od ich reprezentácie a naopak. Podporuje prezentáciu rovnakých údajov rôznymi spôsobmi so zmenami vykonanými v jednej reprezentácii zobrazenej vo všetkých.
Nevýhody	Môže zahŕňať dodatočný kód a zložitosť kódu, keď sú dátový model a interakcie jednoduché .

# Organizácia Model-View-Controller



# (1) Organizácia systému



- ✧ Odráža základnú stratégiu, ktorá sa používa na štruktúrovanie systému.
- ✧ Najčastejšie sa používajú tri organizačné štýly:
  - Štýl **zdieľaného úložiska údajov** ;
  - Štýl **zdieľaných služieb a serverov** ;
  - Abstraktný **strojový alebo vrstvený** štýl.

## (1.1) Vrstvená architektúra



- ✧ Používa sa na modelovanie rozhrania podsystémov.
- ✧ Organizuje systém do množiny vrstiev (alebo abstraktných strojov), z ktorých každá poskytuje množinu služieb.
- ✧ Podporuje postupný vývoj podsystémov v rôznych vrstvách. Ked' sa zmení rozhranie vrstvy, ovplyvní to iba susednú vrstvu.
- ✧ Často je to však umelé, aby sa systémy štruktúrovali týmto spôsobom.

# Vzor vrstvenej architektúry



názov	Vrstvená architektúra
<b>Popis</b>	Organizuje systém do vrstiev so súvisiacimi funkciami spojenými s každou vrstvou. Vrstva poskytuje služby vrstve nad ňou, takže vrstvy najnižšej úrovne predstavujú základné služby, ktoré sa pravdepodobne budú používať v celom systéme .
<b>Príklad</b>	Vrstvený model systému na zdieľanie autorských dokumentov uchovávaných v rôznych knižniciach.
<b>Pri použití</b>	Používa sa pri budovaní nových zariadení nad existujúcimi systémami; keď je vývoj rozdelený medzi niekoľko tímov, pričom každý tím je zodpovedný za vrstvu funkčnosti; keď existuje požiadavka na viacúrovňové zabezpečenie.
<b>Výhody</b>	Umožňuje výmenu celých vrstiev, pokiaľ je zachované rozhranie. V každej vrstve môžu byť poskytnuté redundantné zariadenia (napr. autentifikácia), aby sa zvýšila spoľahlivosť systému.
<b>Nevýhody</b>	V praxi je zabezpečenie čistého oddelenia medzi vrstvami často zložité a vrstva vysokej úrovne môže musieť interagovať priamo s vrstvami nižšej úrovne než cez vrstvu bezprostredne pod ňou. Výkon môže byť problémom z dôvodu viacerých úrovní interpretácie požiadavky na službu, ktorá sa spracováva na každej vrstve .

# Všeobecná vrstvená architektúra



User interface

User interface management  
Authentication and authorization

Core business logic/application functionality  
System utilities

System support (OS, database etc.)

## (1.2) Zdieľané úložisko údajov



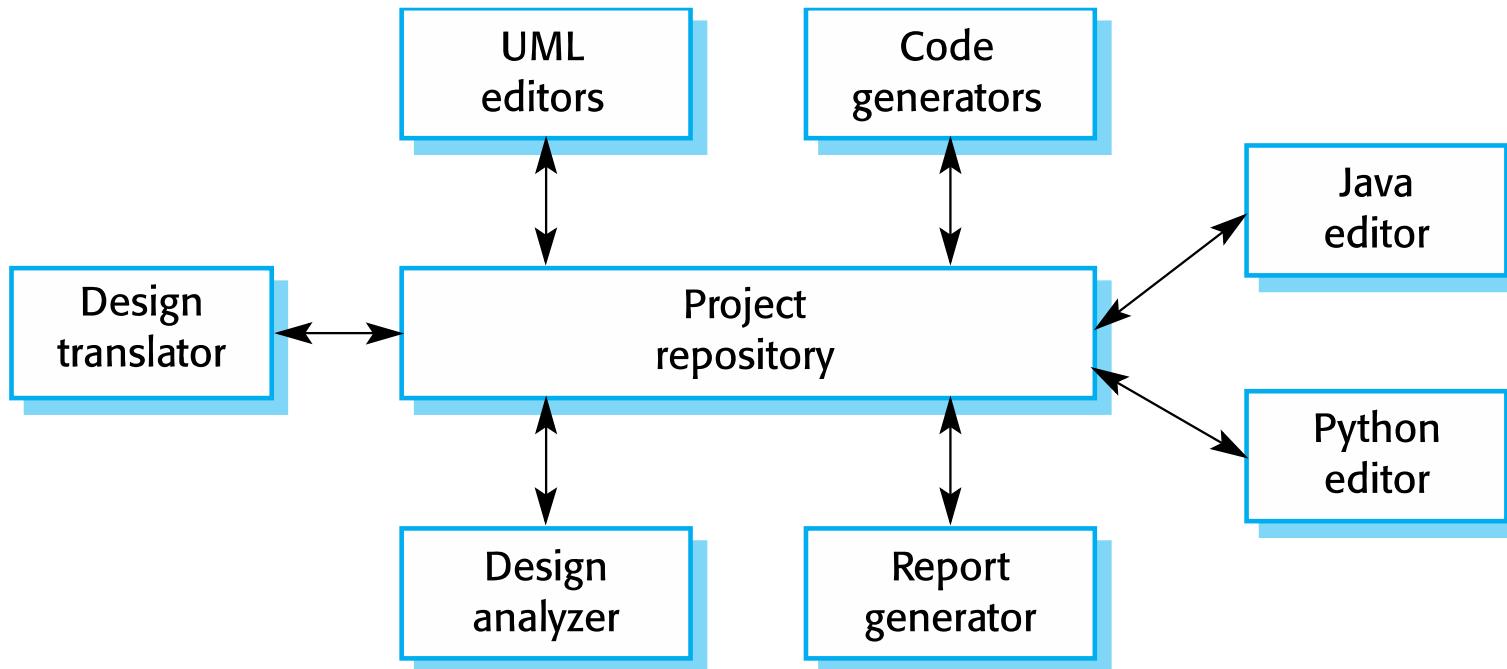
- ✧ Subsystémy si musia vymieňať údaje. To možno vykonať dvoma spôsobmi:
  - **(1.2) Centrálne úložisko:** Zdieľané údaje sa uchovávajú v centrálnej databáze alebo úložisku a môžu k nim pristupovať všetky podsystémy;
  - **(1.3) Distribuované úložiská:** Každý podsystém udržiava svoju vlastnú databázu a údaje explicitne odovzdáva iným podsystémom.
- ✧ Ked' sa majú zdieľať veľké množstvá údajov, najčastejšie sa používa model zdieľania úložiska, čo je účinný mechanizmus zdieľania údajov.

# Vzor centrálneho úložiska



názov	Úložisko
<b>Popis</b>	Všetky dáta v systéme sú spravované v centrálnom úložisku, ktoré je prístupné všetkým komponentom systému. Komponenty neinteragujú priamo, iba prostredníctvom úložiska.
<b>Príklad</b>	Príklad IDE, kde komponenty používajú úložisko informácií o návrhu systému. Každý softvérový nástroj generuje informácie, ktoré sú potom dostupné na použitie inými nástrojmi.
<b>Pri použití</b>	Tento vzor by ste mali použiť, ak máte systém, v ktorom sa generujú veľké objemy informácií, ktoré je potrebné uchovávať na dlhú dobu. Môžete ho použiť aj v systémoch riadených údajmi, kde zahrnutie údajov do úložiska spúšťa akciu alebo nástroj.
<b>Výhody</b>	Komponenty môžu byť nezávislé – nemusia vedieť o existencii iných komponentov. Zmeny vykonané jedným komponentom sa môžu rozšíriť na všetky komponenty. Všetky dáta je možné spravovať konzistentne (napr. zálohovanie v rovnakom čase), keďže sú všetky na jednom mieste.
<b>Nevýhody</b>	Úložisko je jediným bodom zlyhania, takže problémy v úložisku ovplyvňujú celý systém. Môže ísť o neefektívnosť pri organizovaní všetkej komunikácie cez úložisko. Distribúcia úložiska na niekoľko počítačov môže byť náročná .

# Centrálné úložisko pre IDE



## (1.3) Zdieľané služby a servery



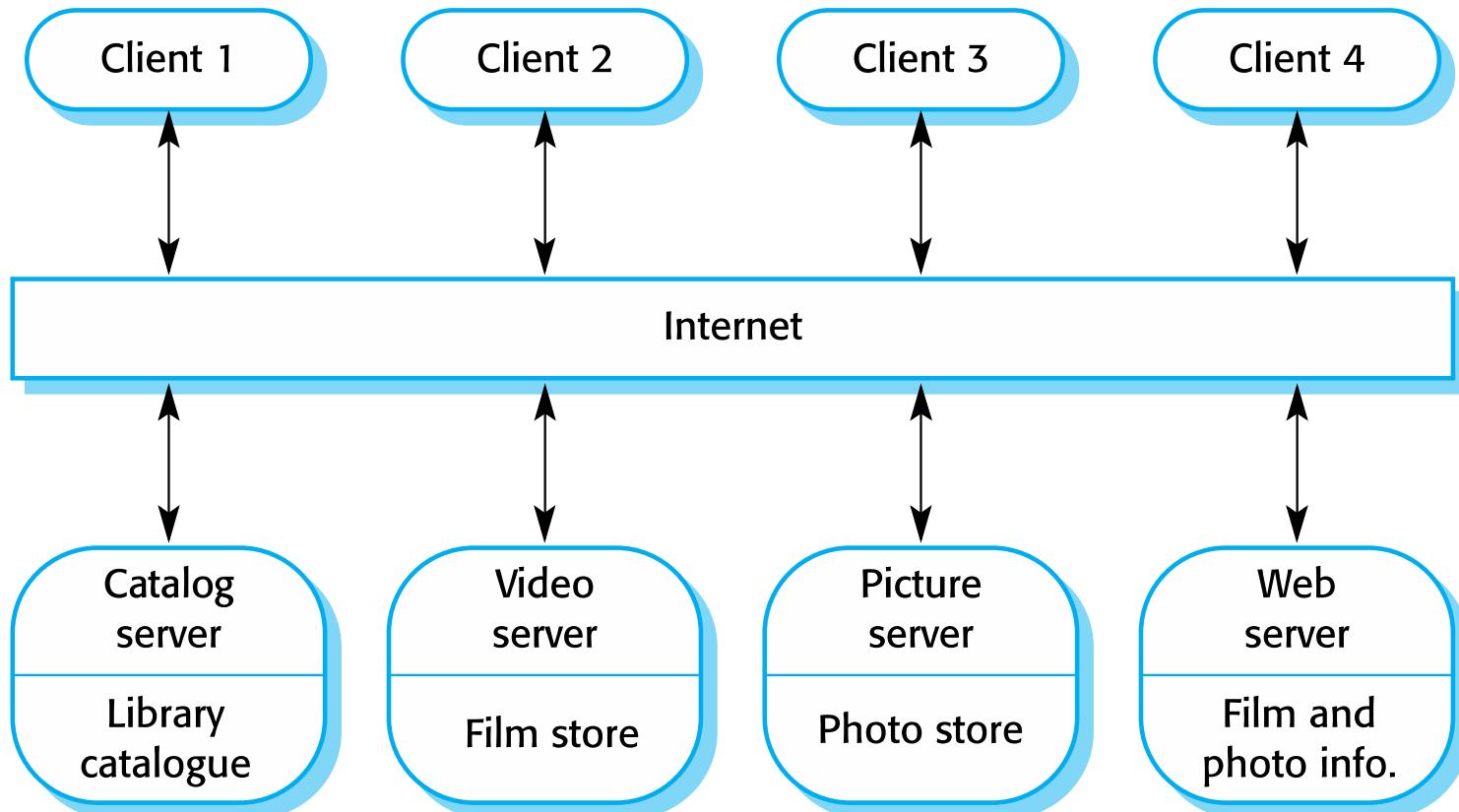
- ✧ Model distribuovaného systému, ktorý ukazuje, ako sú údaje a spracovanie distribuované v rámci rôznych komponentov .
  - Môže byť implementovaný na jednom počítači.
- ✧ Sada samostatných **serverov**, ktoré poskytujú špecifické služby, ako je tlač, správa údajov, atď.
- ✧ Skupina **klientov**, ktorí tieto služby využívajú.
- ✧ **Siet'**, ktorá umožňuje klientom prístup k serverom.

# Vzor klient-server



názov	Klient- server
<b>Popis</b>	V architektúre klient-server je funkčnosť systému organizovaná do služieb, pričom každá služba je poskytovaná zo samostatného servera. Klienti sú používateľmi týchto služieb a pristupujú k serverom, aby ich mohli využívať.
<b>Príklad</b>	Príklad filmovej a video/DVD knižnice organizovanej ako systém klient-server.
<b>Pri použití</b>	Používa sa, keď je potrebné pristupovať k údajom v zdieľanej databáze z rôznych miest. Pretože servery možno replikovať, môžu sa použiť aj vtedy, keď je zaťaženie systému premenlivé.
<b>Výhody</b>	Hlavnou výhodou tohto modelu je, že servery môžu byť distribuované cez sieť. Všeobecná funkcia (napr. tlačová služba) môže byť dostupná pre všetkých klientov a nemusí byť implementovaná všetkými službami.
<b>Nevýhody</b>	Každá služba je jediným bodom zlyhania, ktorý je náchylný na útoky odmietnutia služby alebo zlyhanie servera. Výkon môže byť nepredvídateľný, pretože závisí od siete, ako aj od systému. Ak servery vlastnia rôzne organizácie, môžu nastať problémy so správou .

# Architektúra klient-server pre filmovú knižnicu





## (2) Modulárne štýly rozkladu

- ✧ Štýly rozkladu (pod)systémov na moduly.
  - (Sub)systém je systém ako taký, ktorého prevádzka je nezávislá od služieb poskytovaných inými podsystémami.
  - Modul je komponent systému, ktorý poskytuje služby iným komponentom, ale za normálnych okolností by sa nepovažoval za samostatný systém.
- ✧ Pokryté dva modulárne modely rozkladu
  - **(2.1) Objektový model**, kde je systém rozložený na interagujúce objekty;
  - **(2.2) Model potrubia alebo toku údajov**, kde je systém rozložený na funkčné moduly, ktoré transformujú vstupy na výstupy.

## (2.2) Architektúra potrubia a filtra



- ✧ Funkčné transformácie spracovávajú svoje vstupy a vytvárajú výstupy.
- ✧ Môže byť označovaný ako model potrubia a filtra (ako v prostredí UNIX).
- ✧ Varianty tohto prístupu sú veľmi bežné. Ked' sú transformácie sekvenčné, ide o **dávkový sekvenčný model**, ktorý sa vo veľkej miere používa v systémoch na spracovanie údajov.
- ✧ Nie je vhodný pre interaktívne systémy.

# Vzor potrubia a filtra



názov	Potrubie a filter
<b>Popis</b>	Spracovanie údajov v systéme je organizované tak, že každý komponent spracovania (filter) je diskrétny a vykonáva jeden typ transformácie údajov. Dáta prechádzajú (ako v potrubí) z jedného komponentu do druhého na spracovanie.
<b>Príklad</b>	Príklad potrubného a filtračného systému používaného na spracovanie faktúr.
<b>Pri použití</b>	Bežne sa používa v aplikáciách na spracovanie údajov (dávkové aj transakčné), kde sa vstupy spracúvajú v samostatných fázach, aby sa vytvorili súvisiace výstupy.
<b>Výhody</b>	Lahko pochopiteľné a podporuje opäťovné použitie transformácie. Štýl pracovného toku zodpovedá štruktúre mnohých obchodných procesov. Evolúcia pridávaním transformácií je priamočiara. Môže byť implementovaný ako sekvenčný alebo súbežný systém.
<b>Nevýhody</b>	Formát prenosu údajov sa musí dohodnúť medzi komunikujúcimi transformáciami. Každá transformácia musí analyzovať svoj vstup a zrušiť analýzu výstupu do dohodnutej formy. To zvyšuje réžiu systému a môže znamenať, že nie je možné opäťovne použiť funkčné transformácie, ktoré používajú nekompatibilné dátové štruktúry .

## (3) Štýly ovládania



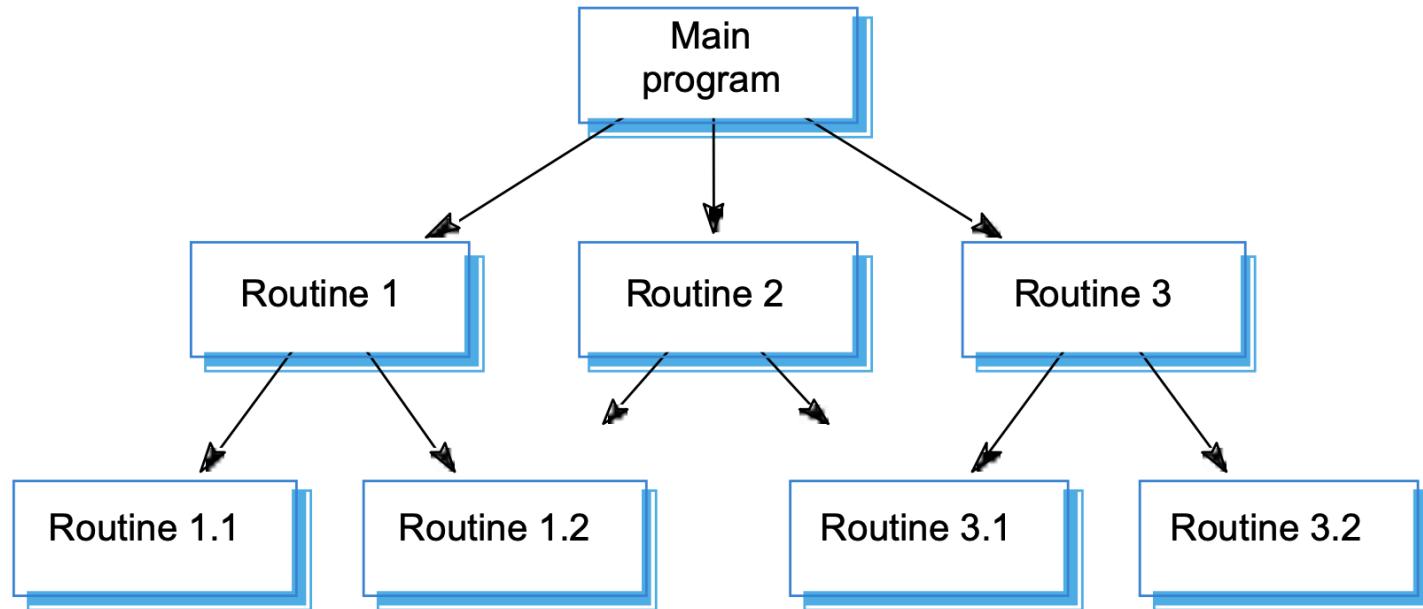
- ✧ Zaoberajú sa riadiacim tokom medzi podsystémami. Na rozdiel od modelu rozkladu systému.
- ✧ (3.1) Centralizované ovládanie
  - Jeden podsystém má celkovú zodpovednosť za riadenie a spúšťa a zastavuje ďalšie podsystémy.
- ✧ (3.2) Ovládanie založené na udalostiach
  - Každý podsystém môže reagovať na externe generované udalosti z iných podsystémov alebo prostredia systému.

# Centralizované ovládanie

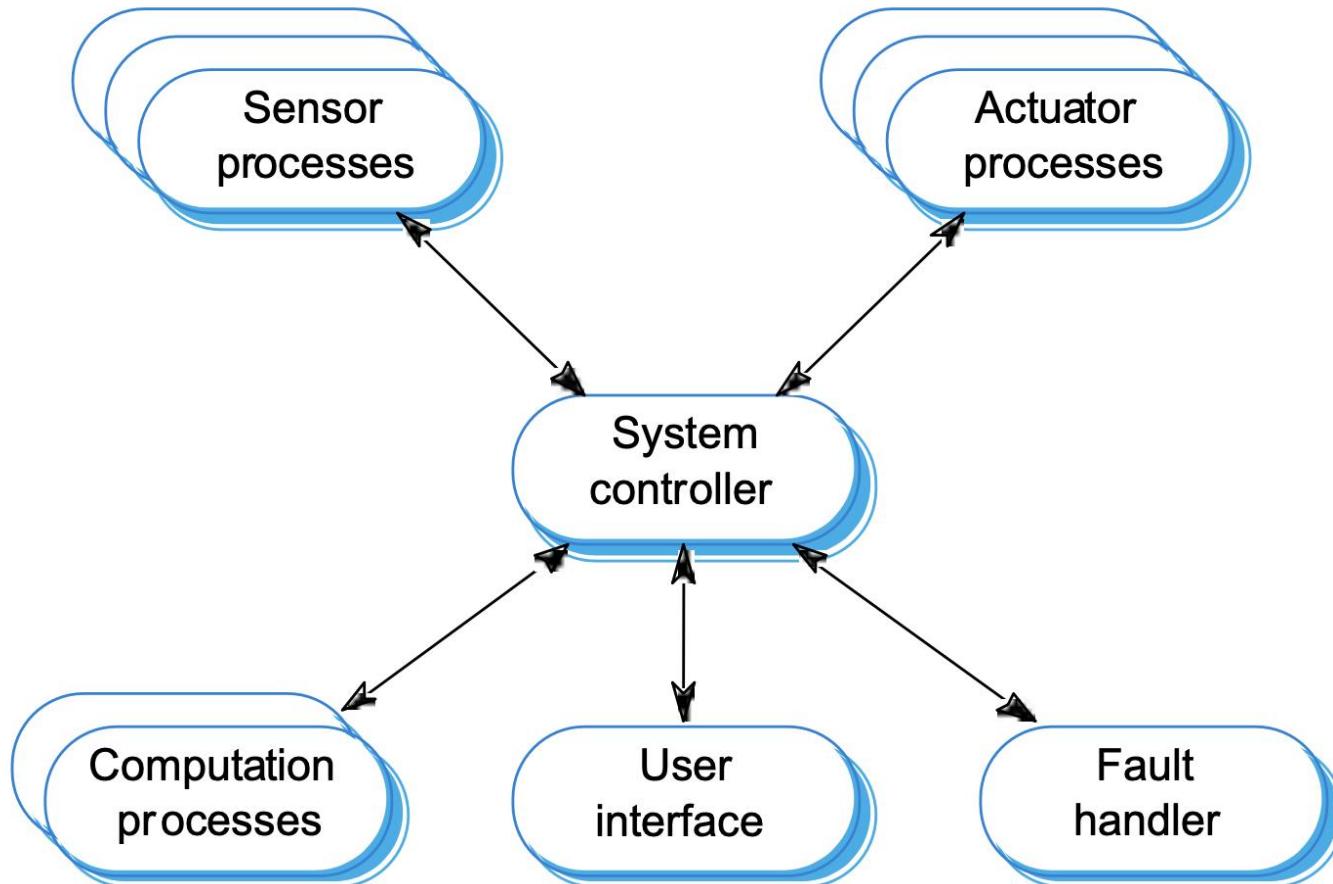


- ✧ Riadiaci subsystém preberá zodpovednosť za riadenie vykonávania iných subsystémov.
- ✧ (3.1.1) Model odovzdania a vrátenia riadenia
  - Model podprogramu zhora nadol, kde riadenie začína na vrchole hierarchie podprogramu a pohybuje sa smerom nadol. Použiteľné pre sekvenčné systémy.
- ✧ (3.1.2) Manažérsky model
  - Použiteľné pre súbežné systémy. Jeden systémový komponent riadi zastavovanie, spúšťanie a koordináciu ostatných procesov systému. Môže byť implementovaný aj v sekvenčných systémoch.

## (3.1.1) Model odovzdania a vrátenia riadenia



## (3.1.2) Manažérsky model



## (3.2) Ovládanie založené na udalostíach

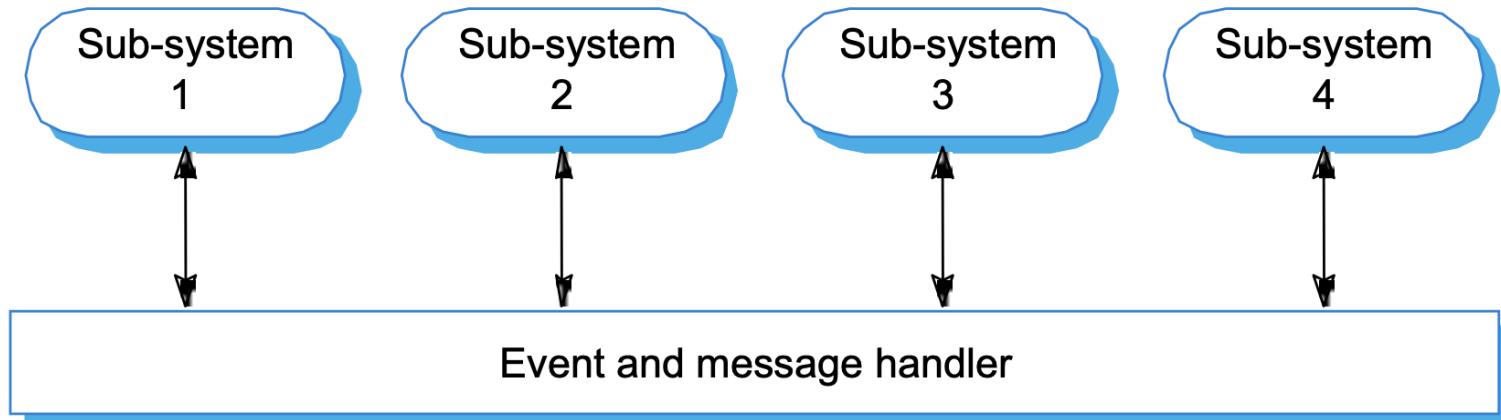


- ✧ Poháňané externe generovanými udalosťami, kde načasovanie udalosti je mimo kontroly podsystémov, ktoré udalosť spracúvajú.
- ✧ Dva hlavné modely riadené udalosťami
  - **(3.2.1) Vysielacie modely** . Udalosť sa vysiela do všetkých podsystémov. Môže tak urobiť akýkoľvek podsystém, ktorý dokáže spracovať udalosť;
  - **(3.2.2) Modely riadené prerušením**. Používa sa v systémoch v reálnom čase, kde sú prerušenia detegované obsluhou prerušenia a odovzdané nejakému inému komponentu na spracovanie.
- ✧ Medzi ďalšie modely riadené udalosťami patria tabuľky a produkčné systémy.



## (3.2.1) Vysielací model

- ✧ Efektívne pri integrácii podsystémov na rôznych počítačoch v sieti.
- ✧ Podsystémy registrujú záujem o konkrétnu udalosť. Ked' k tomu dôjde, riadenie sa prenese na podsystém, ktorý môže udalosť zvládnuť.
- ✧ Riadiaca politika nie je vložená do obsluhy udalosti a správy. Podsystémy rozhodujú o udalostach, ktoré ich zaujímajú.
- ✧ Podsystémy však nevedia, či a kedy sa udalosť spracuje.



## (3.2.2) Riadenie prerušením



- ✧ Používa sa v systémoch v reálnom čase, kde je nevyhnutná rýchla reakcia na udalosť.
- ✧ Existujú známe typy prerušení s obsluhou definovanou pre každý typ.
- ✧ Každý typ je spojený s umiestnením pamäte a hardvérový prepínač spôsobí prenos do jeho obsluhy.
- ✧ Umožňuje rýchlu odozvu, ale zložité programovanie a náročné overenie.

# Systém riadený prerušením

