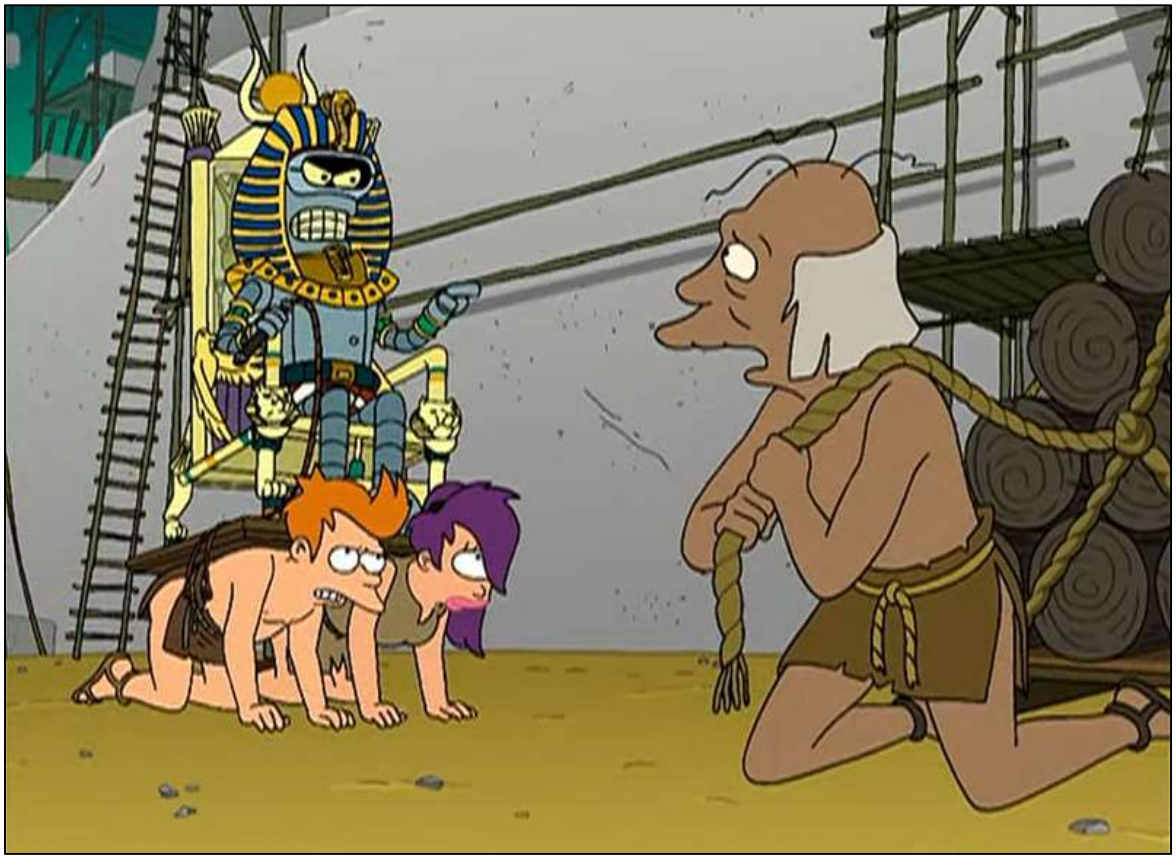
Rozšiřující návrhové vzory pro organizaci práce a komunikaci

Vzory pro organizaci práce

- **situace**
  - službu příliš složitá
  - vhodné/nutné implementaci rozdělit do více spolupracujících komponent
- **cíle**
  - každá komponenta - jasně definované povinnosti
  - základní strategie pro poskytování služby by neměla být rozprostřena napříč příliš velkým množstvím komponent
- **principy**
  - separation of concerns (oddělení zodpovědností)
  - separation of policy and mechanism (oddělení „co“ a „jak“)
  - divide and conquer (rozděl a panuj)
- **příklady**
  - vzor chain of responsibility
  - vzor command
  - vzor mediator
  - ?

Master-Slave



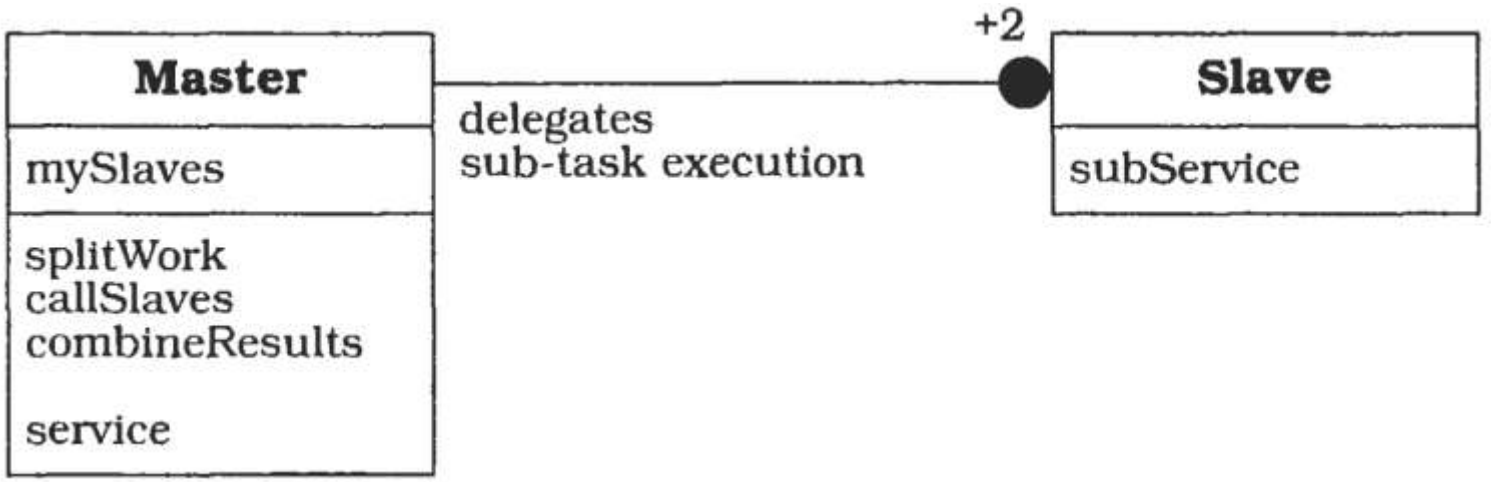
Futurama, A Pharoh to Remember, © Comedy Partners

Problém

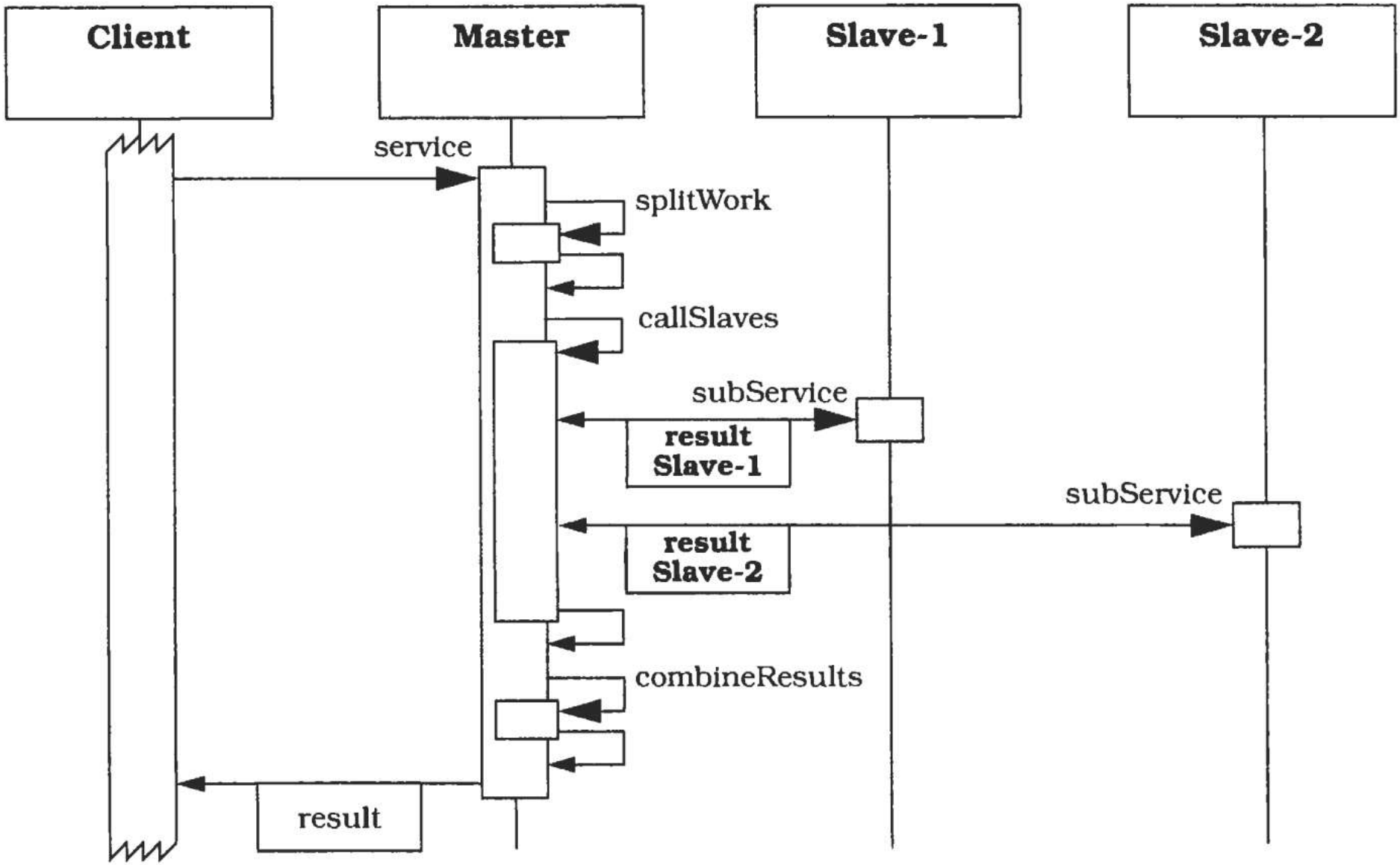
- **úloha**
  - aproximace řešení problému obchodního cestujícího tak, že vybereme náhodně a nezávisle na sobě dostatečně velké množství tras, ze kterých vybereme tu nejkratší
- **strategie řešení**
  - aplikujeme princip rozděl a panuj
- **požadavky na řešení**
  - klient by měl být odstíněn od toho, jak je práce na výpočtu organizována
  - jak klient, tak samotné zpracování podúloh by nemělo záviset na algoritmu pro rozdělení zadání na podúlohy či sestavení výsledku z dílčích výsledků podúloh
  - je užitečné, pokud můžeme užít různé, ale sémanticky identické implementace pro zpracování podúloh
  - zpracování podúloh je v některých typech zadání vhodné koordinovat

Řešení se vzorem master-slave

- **koordinační komponenta (master)**
  - poskytuje klientovi rozhraní pro zadání úlohy
  - postará se o rozdělení práce do přibližně stejně náročných podúloh
  - spravuje podřízené komponenty a distribuuje mezi ně podúlohy
  - odstartuje a řídí výpočet v podřízených komponentách
  - odvodí z výsledku jejich práce konečný výsledek, který obrdrží klient
- **podřízené prvky (slaves)**
  - poskytují kooridnátorovi rozhraní pro zadání podúlohy
  - provádí samotný výpočet na poskytnutých datech

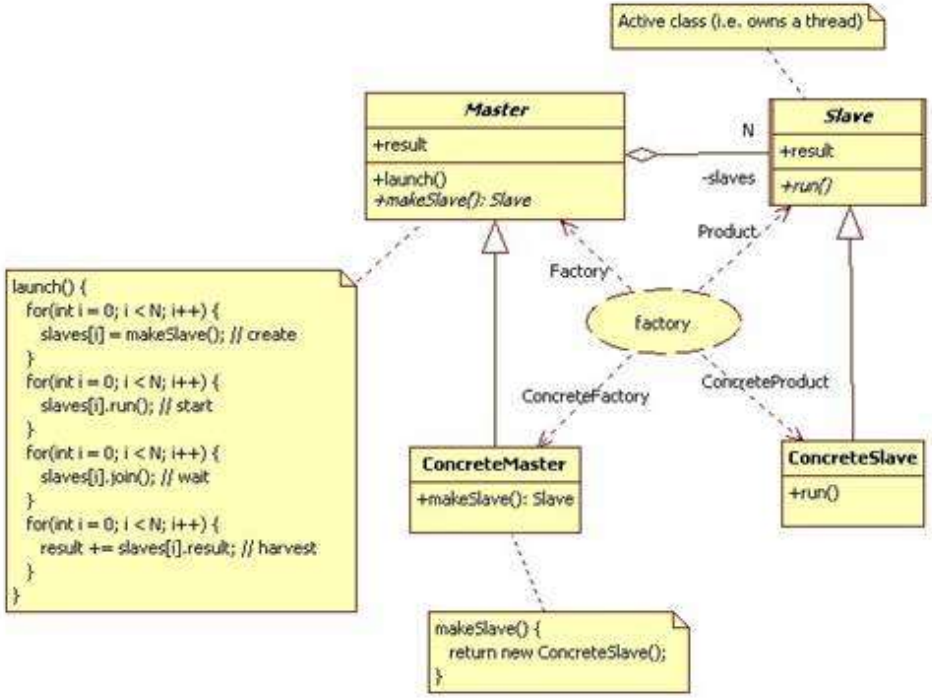


Postup zpracování



Otázky k implementaci

- **Jak rozdělit na podúlohy?**
- **Jak sloučit výsledky?**
- **Jak budou otroci přistupovat k datům / vracet výsledky?**
  - dostanou přímo jako argument / jako návratovou hodnotu
  - přistupují ke společnému úložišti / zapisují tam
  - nutno zvolit s ohledem na podmínky (síť, množství uzlů, druh úlohy,...)
- **Jak se ubránit chybám a výpadkům?**



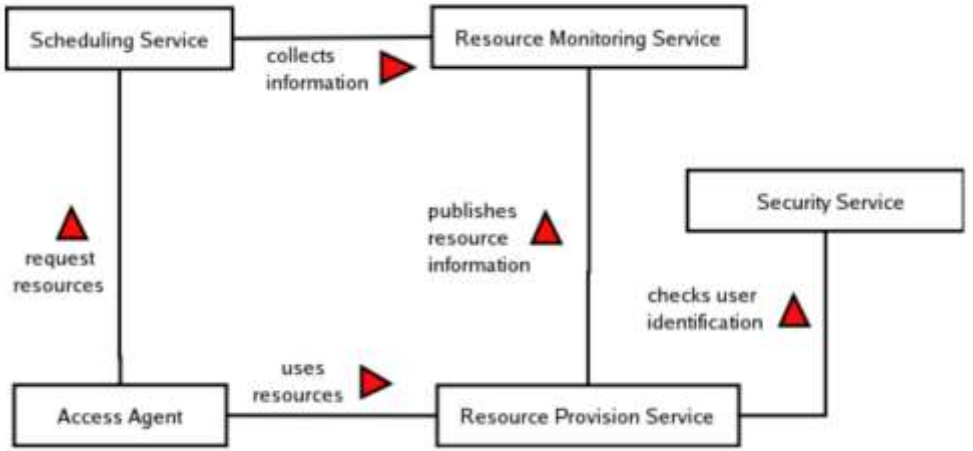
Varianty

- **pro zvýšení odolnosti vůči chybám**
  - všichni otroci dostanou stejné zadání
  - master zůstává ale pořád pouze jeden!
- **pro paralelní výpočty**
  - každý otrok dostane za úkol spočítání části úlohy
  - master musí počkat s odevzdáním výsledku na doběhnutí podúloh
- **pro zvýšení přesnosti výsledku**
  - otroci mají různé způsoby implementace
  - všichni dostanou stejné zadání
  - master konečný výsledek odvodí na základě nějaké strategie (nejvíce shod, průměr)
- **otroci jsou procesy**
  - master s otrokem nakládá pomocí proxy
- **otroci jsou ve vláknech**
- **s koordinací otroků**
  - distribuovaná
  - skrze mastera

Výhody a nevýhody

- **snadná zaměnitelnost a rozšiřitelnost částí**
  - můžeme změnit implementaci otroka pouze s malými změnami mastera, klient je od takovéto změny zcela odstíněn
  - podobně u mastera, pokud zavádíme způsob rozdělení úlohy mezi otroky a způsob výpočtu konečného výsledku pomocí vzoru strategie
- **oddělení zodpovědností**
  - jak kód klienta, tak otroků nemusí řešit dělení na podúlohy, delegování na otroky, sběr a výpočet konečného výsledku, ani výpadky či pojištění proti chybám
- **efektivita**
  - paralelní zpracování úloh může u vhodného typu úloh podstatně urychlit dodání výsledku
- **není vhodné pro všechny typy úloh**
  - nemusí se vyplatit režie
- **závislost na infrastruktuře**
  - na hardware i software
- **náročnost implementace**
  - může být těžké převést do nezávislých podúloh
  - typické problémy s paraleleními výpočty
  - obsluha chyb

Souvislosti



- **grid**
  - podobný vzor
  - pro heterogenní prostředí
- **problémy s pojmenováním**
  - 2003: Los Angeles County požaduje po výrobcích a dodavatelech, aby přestali tato slova užívat „na základě rozmanitosti místních kultur a ohledu k nim“
  - 2004: Global Language Monitor (nejvíce politicky nekorektní slovo roku)
  - květen 2014: framework Django (leader/follower, posléze primary/replica).
  - červen 2014: Drupal (replica, zdůvodněno zavedenou praxí ve firmách jako IBM, Microsoft, Amazon a další)

Obrázek z DE CAMARGO, Raphael Y., Andrei GOLDCHLEGER, Marcio CARNEIRO a Fabio KON. Grid: An architectural pattern

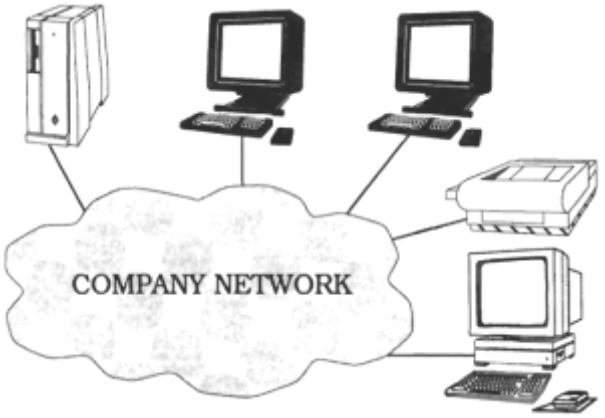
Vzory pro komunikaci

- **situace**
  - máme distribuovanou aplikaci (ta se sestává z uzlů, které spolu musí komunikovat)
- **cíle**
  - chceme se vyhnout tomu, aby použitý způsob komunikace byl těsně svázán s logikou aplikace a aby šel případně snadno nahradit
  - chceme, aby data která mají k dispozici účastníci, byla konzistentní
- **principy**
  - encapsulation (zapouzdření)
    - skrývání detailů podpůrného komunikačního mechanismu od jeho uživatelů
  - location transparency (transparentnost umístění)
    - užití jména k identifikaci síťového zdroje, které je nezávislé na jeho fyzickém umístění
- **příklady**
  - ?
  - ?
  - ?

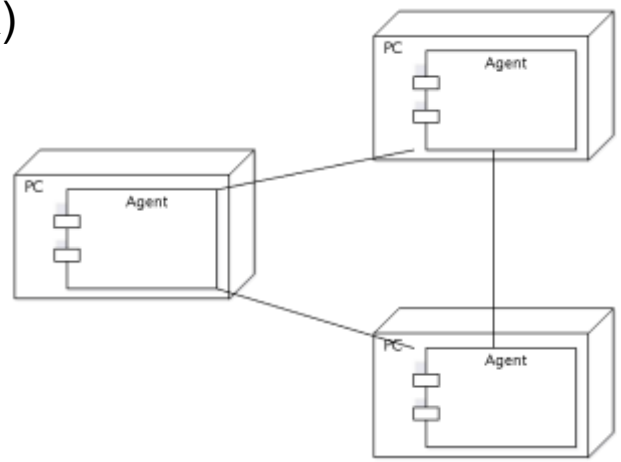
Forwarder-Receiver

Futurama © Comedy Partners

Problém



- **úloha**
  - máme infrastrukturu pro správu sítě, která je kromě ostatních komponent tvořena procesy, běžícími na platformě Java
  - tyto agenty mají na starosti pozorování a monitorování událostí a zdrojů v síti
  - navíc dovolují administrátorům změnit a řídit chování sítě (například překonfigurováním routovacích tabulek)
  - pro výměnu informací a rychlou propagaci změn jsou agenty mezi sebou propojeny peer-to-peer, podle potřeby se tedy chovají jako server či klient
  - protože infrastruktura musí podporovat rozličný hardware i software, komunikace mezi peery nesmí záležet na určitém způsobu meziprocesové komunikace



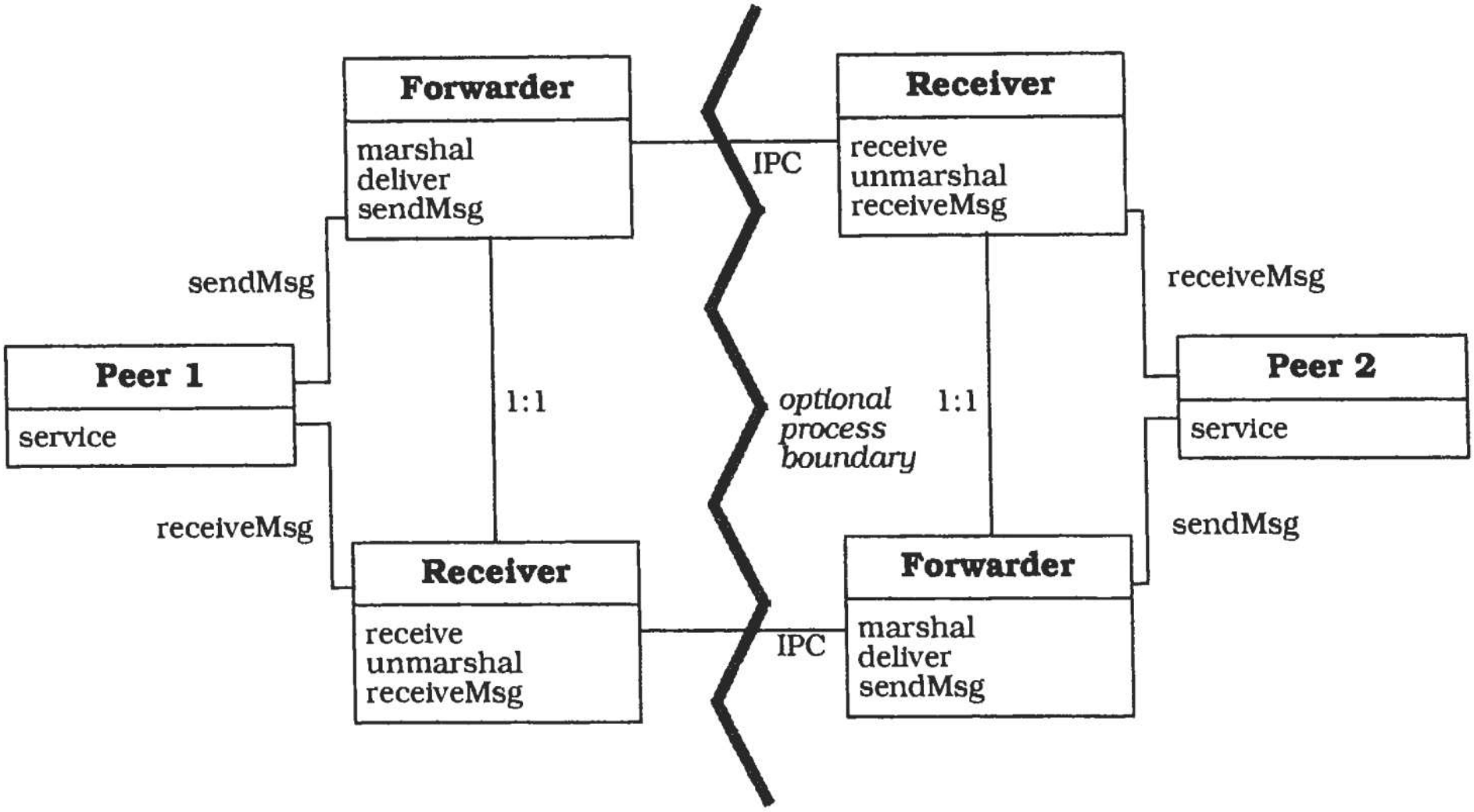
- **požadavky na řešení**
  - mělo by být možné zaměnit komunikační mechanismy
  - odesílateli by mělo stačit znát pouze jména příjemců
  - komunikace mezi peery by neměla mít zásadní dopad na výkon

Zdroj diagramu s agenty: MFF UK, předmět Návrhové vzory, prezentace POSA, Michal Folk, 2010

Řešení se vzorem forwarder-receiver

- **peer**
  - obsahuje aplikační logiku
  - má potřebu komunikovat s ostatními peery (zná jejich jména)
    - zprávy (ať už požadavky či potvrzení) zasílá přes forwardera a přijímá přes receivera
- **forwarder**
  - zasílá zprávy napříč procesy
  - poskytuje obecné rozhraní, jež je abstrakcí konkrétního způsobu meziprocesové komunikace
  - implementuje zabalení a odeslání zprávy
  - pamatuje si převodní tabulku jmen na fyzické adresy
    - aby mohl podle jména ve zprávě určit, kam má zprávu poslat
    - naopak do každé zprávy uvede jméno obhospodařovaného peera, aby mu příjemce mohl zaslat odpověď
- **receiver**
  - odpovídá za příjem zpráv
  - implementuje rozbalení a příjem zprávy

Znázornění


Forwarder
marshal
deliver
sendMsg
Receiver
receive
unmarshal
receiveMsg
IPC
sendMsg
receiveMsg
Peer 1
service
1:1
optional
process
boundary
1:1
Peer 2
service
receiveMsg
sendMsg
Receiver
receive
unmarshal
receiveMsg
IPC
Forwarder
marshal
deliver
sendMsg

Poznámky k implementaci

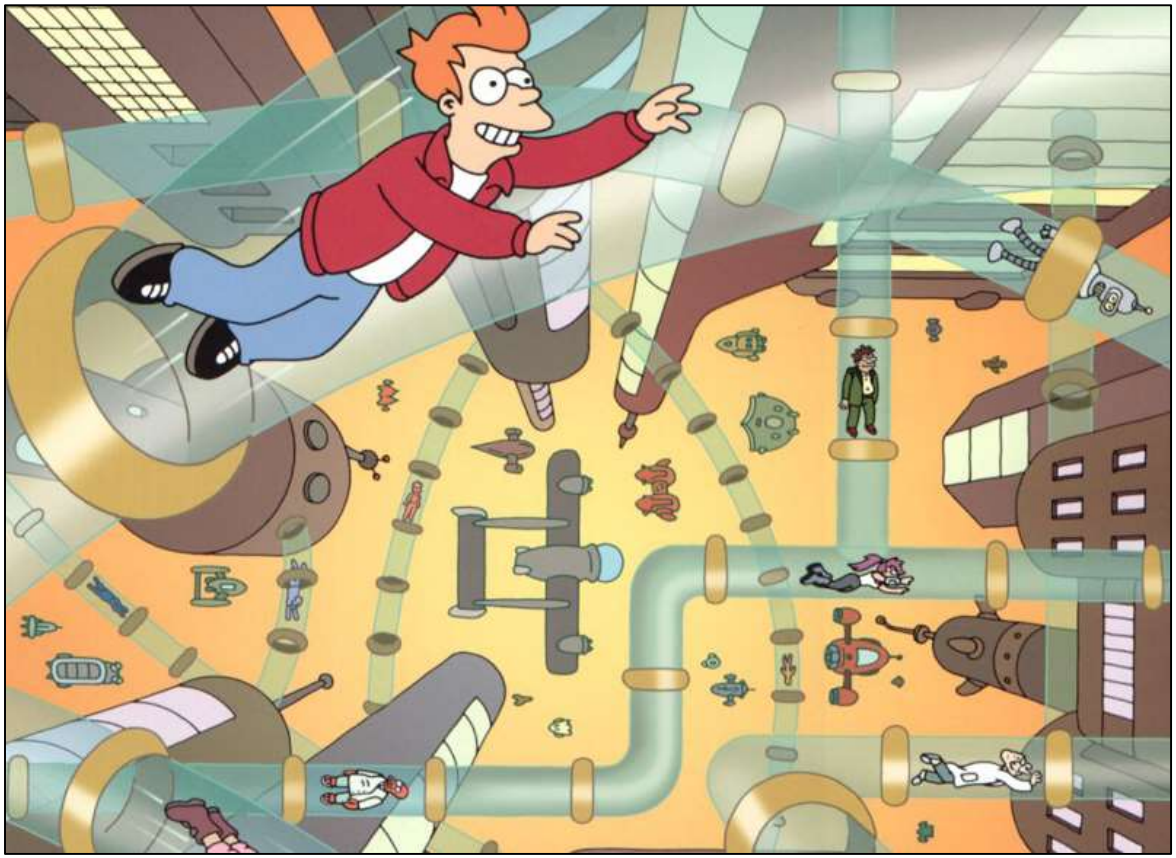
- **adresování**
  - mělo by být pravidelné
  - adresa nemusí nutně označovat individuální uzel, ale skupinu
    - lze zavést dokonce hierarchii skupin
- **obsluha vypršení času a selhání komunikace**
  - peer může dát forwarderu a receiveru odpovědi čas, který mu připadá vhodný
    - nebo ponechat nastavení v režii forwarderu
  - má se forwarder pokoušet opakovat odeslání, či hned vyvolat výjimku?
    - rozhodutí silně závisí na užitém způsobu komunikace
- **implementace receiveru**
  - blokující
  - neblokující
- **podoba překladové tabulky**
  - sdílená
  - v každém forwarderu

Výhody a nevýhody, varianta

- **efektivita**
  - každý forwarder zná fyzickou adresu receiveru, a tedy nemusí ji nikde dohledávat
  - přidání nové vrstvy má zanedbatelné nároky ve srovnání s vlastní komunikací
- **zapouzdření meziprocesové komunikace**
  - změna podkladového mechanismu komunikace nemá vliv na peery
- **chybí podpora pro pružnou změnu rozložení peerů za běhu**
  - dá se řešit rozšířením o následující vzor

- **forwared-receiver bez překladu jmen na adresy**
  - obětujeme nezávislost na fyzických adresách za zlepšení výkonu

Client-Dispatcher-Forwarder



Futurama © Comedy Partners

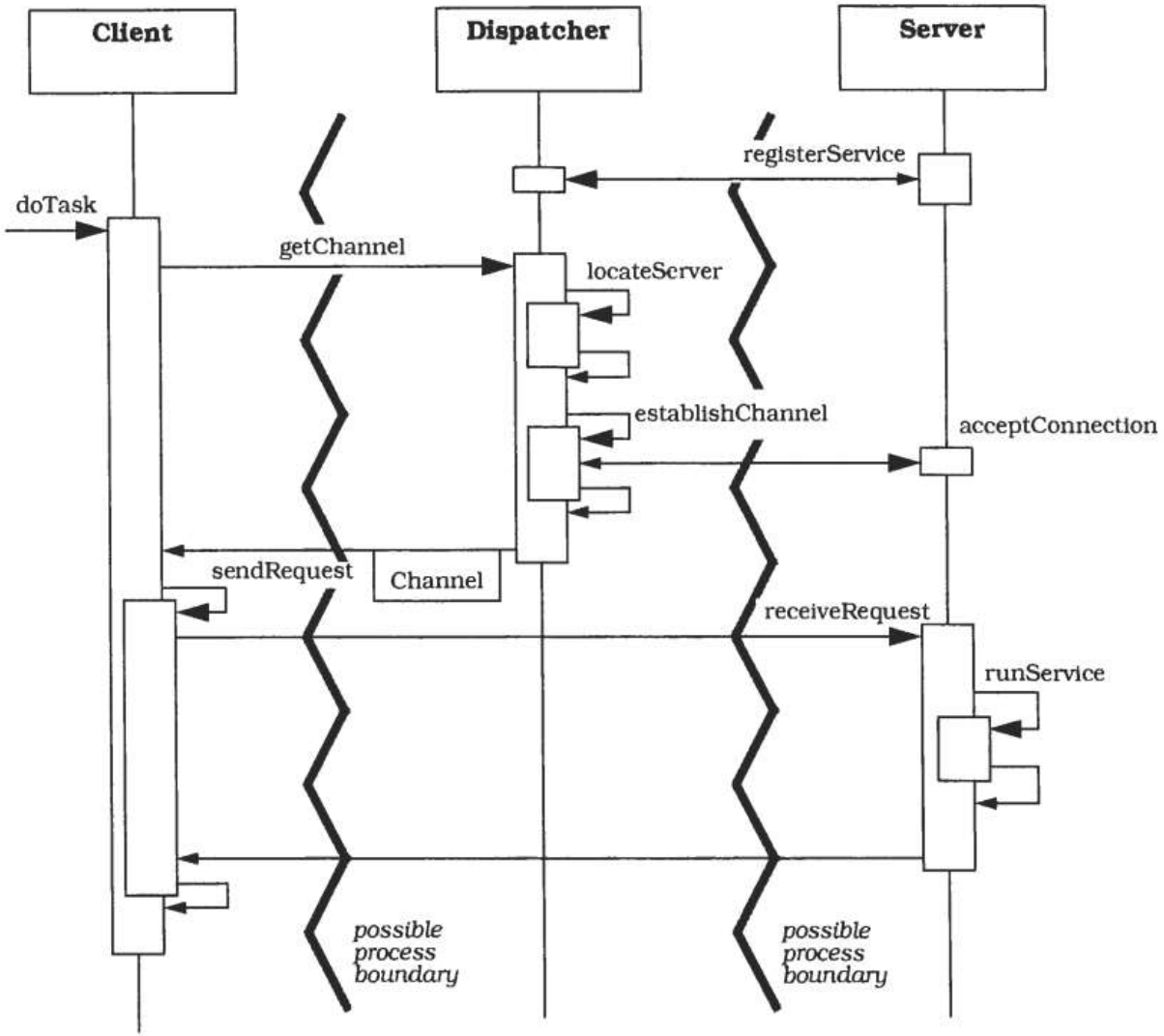
Problém

- **úloha**
  - knihovna má centrálu a jedno z tématických oddělení na jednom místě
    - sdílí místní infrastrukturu
  - ostatní oddělení se nachází na různých dalších místech
  - každé oddělení vede záznamy a poskytuje k nim přístup samo
  - uživatel chce hledat v centrále publikace nacházející se v různých tématických odděleních
- **požadavky na řešení**
  - klient by měl být schopen používat službu bez ohledu na to, v jakém místě se nachází její poskytovatel
  - kód pro využívání služby by měl být oddělen od kódu pro zajištění spojení s poskytovatelem služby

Řešení se vzorem client-dispatcher-server

- **dispatcher**
  - poskytuje službu, jež umožňuje klientovi používat pro identifikaci serverů jejich jména a nikoli fyzická umístění (transparentnost umístění)
  - navíc se stará o vytvoření komunikačního kanálu mezi klientem a serverem
- **server**
  - poskytuje služby klientům
  - registruje se či je registrován pod svým jménema adresou u dispatchera
  - může se nacházet na stejném stroji jako klient, ve stejné místní síti či být dostupný přes Internet (nutná volba vhodného způsobu komunikace pro optimální výkon)
- **klient**
  - spoléhá na dispatchera s nalezením serveru a pro vytvoření spojení s ním
    - nadále pak pracuje pouze s kanálem k serveru, který mu dispatcher poskytl
  - může si za běhu prohodit roli se serverem

Znázornění průběhu



Varianty

- **distribuované dispatchery**
  - zavádí se více dispatcherů
  - dispatcher se po žádosti od svého klienta spojí s dispatcherem cíle, který vytvoří kanál, který je přes původní dispatcher předán klientovi
  - klient může také kontaktovat dispatchera serveru přímo, tím ale přicházíme o transparentnost umístění
- **komunikace spravována klientem**
  - dispatcher nevytváří spojení se serverem, ale pouze předá klientovi jeho adresu
- **s více způsoby komunikace**
  - servery se navíc registrují s podporovanými druhy komunikace
- **client-dispatcher-service**
  - dispatcher si pamatuje mapování služeb na implementující servery
  - na žádost o připojení dodá nějaký server, který službu podporuje
  - pokud je dodaný server nedostupný, dispatcher se může pokusit dodat jiný vhodný
- **kombinace se vzorem forwarder-receiver**

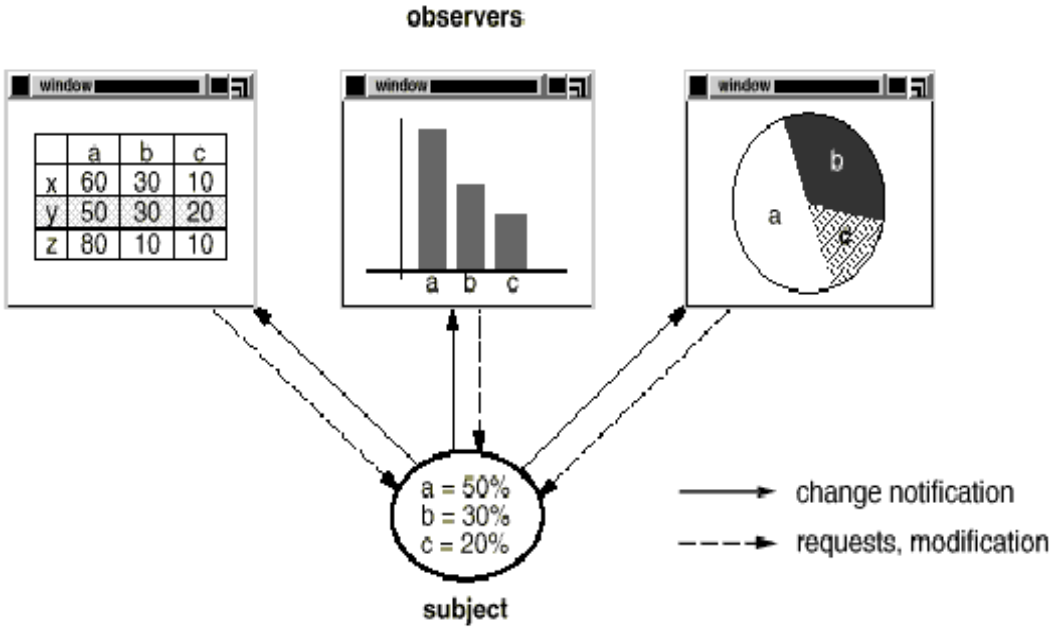
Výhody a nevýhody

- **snadná zaměnitelnost serverů, flexibilita a odolnost proti selhání**
  - servery se mohou registrovat i odregistrovat z dispatchera za běhu
- **transparentnost umístění a přesunu**
  - klienti pro vyhledání serveru nepotřebují fyzickou adresu
  - servery lze i přesouvat, pokud nejsou zrovna využívány
- **zhoršená efektivita komunikace**
  - nutná registrace serverů
  - dohledávání a vytváření explicitního spojení

Publisher-Subscriber

Připomenutí

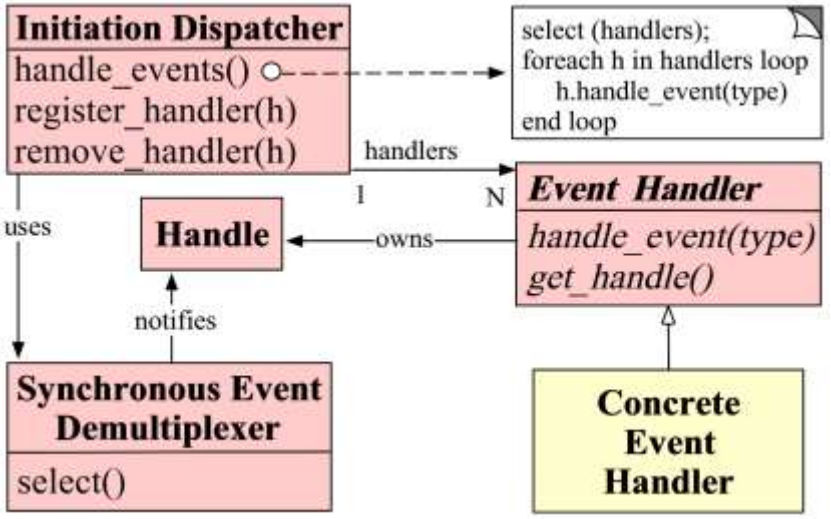
- **vzor publisher-subscriber je jiné označení pro vzor observer**
  - publisher je subject podle GoF
  - subscriber je observer



Zdroj obrázku: MFF UK, předmět Návrhové vzory, prezentace Obeserver pattern, Miroslav Baron

Gatekeeper

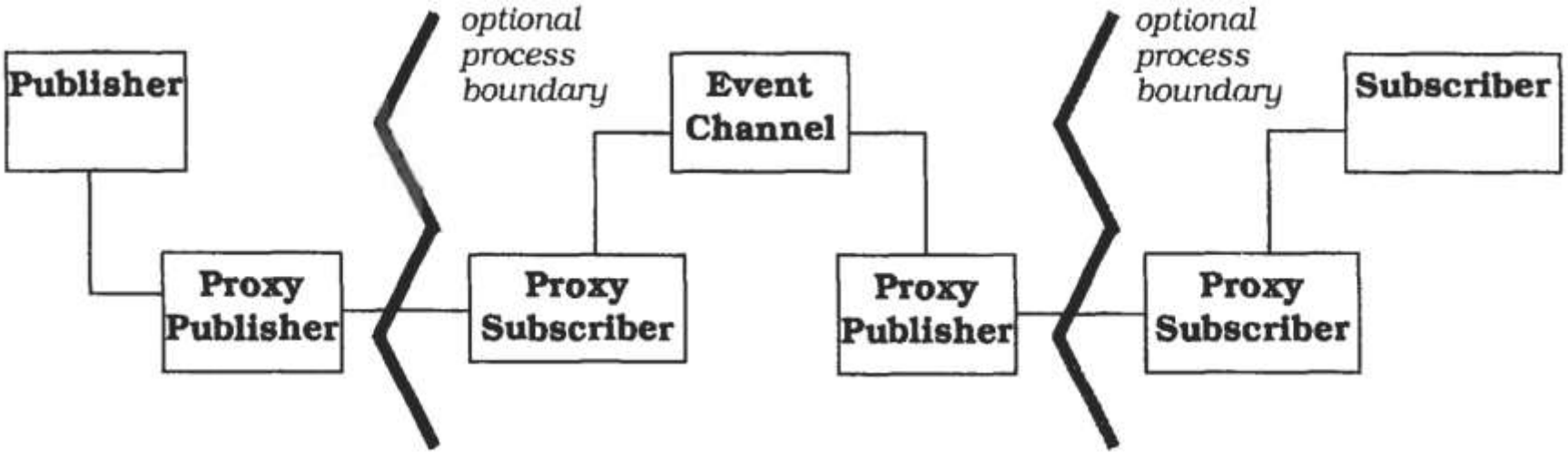
- **jedna z variant pro distribuované systémy**
- **průběh rozdělen do dvou procesů**
  - proces zasílající zprávu o změně
  - přijímající proces obsahující samotný gatekeeper (který je aplikací vzoru reactor)
- **vzor Reactor**
  - zpracovává souběžně doručené požadavky z **vícero vstupů** v **jednom vláknu**
  - užívá synchronní demultiplexor (v Unixu systémové volání select(1))
    - ve smyčce se testuje, dokud nějaký tzv. handle (spojený s nějakým systémovým zdrojem, například síťovým spojením) nelze synchronně zpracovat bez zablokování
  - zpracování v handleru není preemptivní



Obráek z SCHMIDT, Douglas C. Reactor: An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events

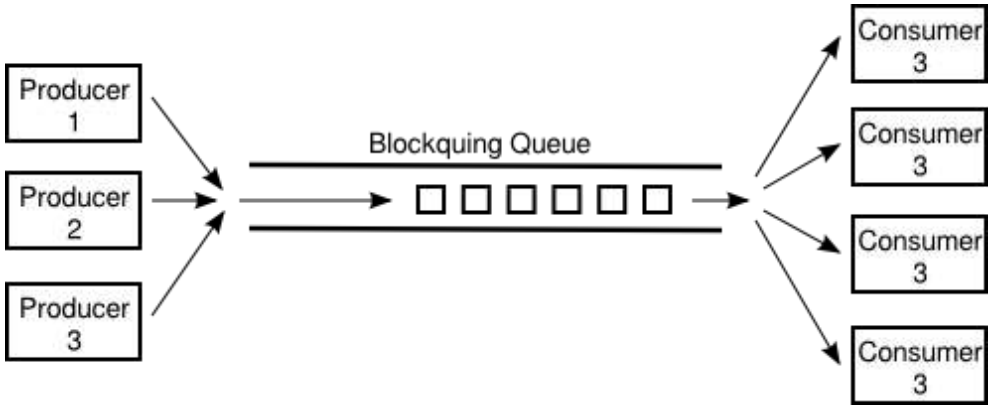
Event Channel

- **je jednou z variant vzoru publisher-subscriber pro distribuované systémy**
- **zpřetrhává závislosti mezi publisherem (těch může být i více) a subscribery**
- **event channel**
  - umístěn mezi publishery a subscribery
  - vůči publisherovi se chová jako subscriber a naopak
  - proxy umožňují maskovat skutečnou existenci všech tří druhů komponent v různých procesech
  - může být obohacen o buffer, filtry, zřetězen s dalšími kanály



Ve stylu Producer-Consumer

- **opět silně rozvázaná spolupráce publishera (producer) a subscribera (consumer)**
- **producer může do společného bufferu publikovat dle libosti, dokud není plný, pak je zastaven**
- **naopak consumer se zastaví, pokud je buffer při pokusu o čtení dat prázdný**
- **oproti P-S je poměr producerů a consumerů typicky 1:1**
- **pro více producerů a consumerů lze implementovat pomocí event channelu**



Obrázek z DAVID J. ECK. Programming: introduction to programming using Java. San Francisco, Calif: SoHo Books, 2007. ISBN 978-144-1419-767.

Shrnutí

- **vzory pro organizaci práce**
  - Master-Slave
- **vzory pro komunikaci**
  - Forwarder-Receiver
  - Client-Dispatcher-Receiver
  - Publisher-Subscriber
    - Gatekeeper
    - Event Channel
    - ve stylu Producer-Consumer

Zdroje

- BUSCHMANN, Frank.
  Pattern-oriented software architecture: a system of patterns.
  New York: Wiley, c1996. ISBN 0471958697. (+ obrázky pokud není uvedeno jinak)
- DE CAMARGO, Raphael Y., Andrei GOLDCHLEGER, Marcio CARNEIRO a Fabio KON.
  Grid: An architectural pattern.
  The 11th Conference on Pattern Languages of Programs (PLoP'2004). 2004, 2004(1), 1-17.
- SCHMIDT, Douglas C.
  Reactor: An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events.
  Pattern Languages of Programs. 1994, 1-11.