الگوها و تاکتیک‌های معمارانه

فریدون شمس

فهرست مطالب

- تاکتیک‌ها و الگوهای معمارانه
- الگوهای ماژول
- الگوهای مؤلفه و اتصال
- الگوهای تخصیص
- ارتباط میان الگو و تاکتیک
- استفاده از تاکتیک‌ها در کنار هم

تاکتیک‌ها و الگوهای معمارانه

- الگوها و تاکتیک‌های معمارانه راه‌های دستیابی به ساختارهای خوب طراحی است. بنابراین از آن‌ها می‌توان بارها استفاده کرد
- الگوی معمارانه:
  - بسته‌ای از تصمیمات طراحی متداول است
  - دارای ویژگی‌های شناخته شده است که امکان استفاده مجدد از آن را فراهم می‌کند
  - کلاسی از معماری را تعریف می‌کند
- تاکتیک‌ها بلوک‌های سازنده معماری هستند که الگوهای معماری از آن‌ها ساخته می‌شوند
  - تاکتیک‌ها مانند اتم و الگوها همانند مولکول هستند
- بیشتر الگوها از چندین تاکتیک مختلف تشکیل شده‌اند

الگوی معمارانه

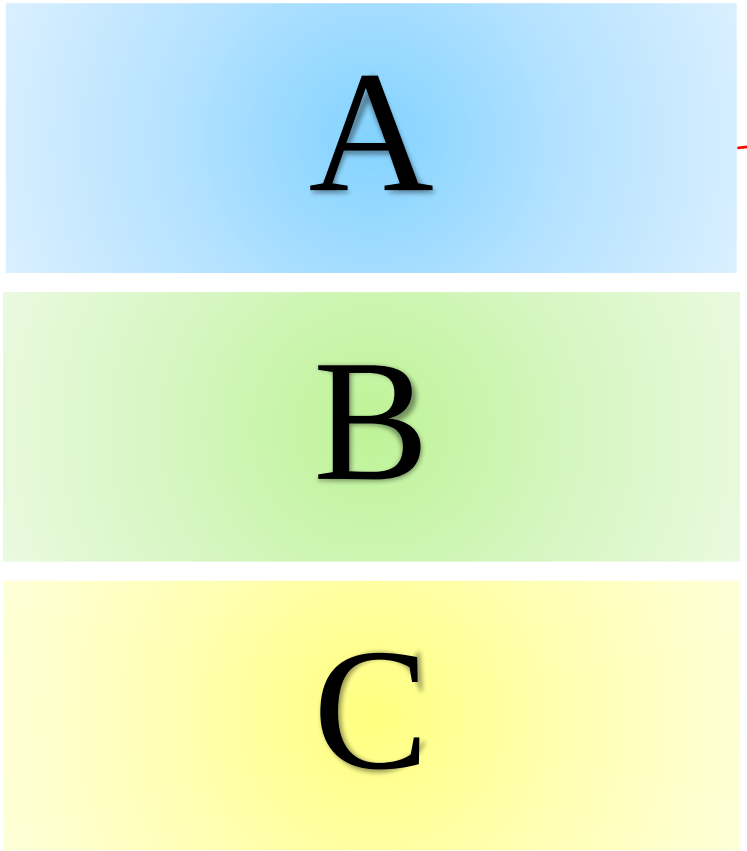
- الگوی معمارانه ارتباطی میان اجزای زیر ارتباط برقرار می‌کند:
  - زمینه (*Context*): شرایطی متداول و تکرار شونده در جهان که منجر به ایجاد مسئله‌ای می‌شود
  - مسئله (*Problem*): یک مسئله، قابل تعمیم، که در یک زمینه مشخص به وجود می‌آید
  - راه‌حل (*Solution*): یک راه‌حل معمارانه موفق، در سطح انتزاع مناسب، برای مسئله. راه‌حل یک الگو توسط موارد زیر تعیین و تعریف می‌شود:
    - مجموعه‌ای از نوع عناصر (مانند انباره‌های داده، فرایندها و اشیا)
    - مجموعه‌ای از سازوکارهای تراکنش یا اتصال‌دهنده‌ها (مانند فراخوانی متدها، رخدادها، گذرگاه پیام)
    - طرح توپولوژیک مؤلفه‌ها
    - مجموعه‌ای از محدودیت‌های معنایی مربوط به توپولوژی، رفتار عناصر و سازوکارهای تعامل
- {زمینه، مسئله، راه‌حل} به منزله قالبی برای مستندسازی الگوها به شمار می‌روند

الگوهای ماژول

الگوی لایه‌ای (*Layer*)

- زمینه:
  - تمامی سیستم‌های پیچیده نیازمند توسعه و تکامل سیستم به صورت بخش‌های مستقل هستند. بنابراین جداسازی دغدغه‌های سیستم به طور واضح و مستند شده یک نیاز اساسی توسعه‌دهندگان سیستم است، به طوری که بتوان ماژول‌های سیستم را به صورت مستقل توسعه و نگهداری نمود
- مسئله:
  - نرم‌افزار باید به گونه‌ای تقسیم‌بندی شود که ماژول‌ها به صورت جداگانه و با حداقل ارتباط با دیگر بخش‌ها توسعه پیدا کرده، تکامل یافته و از قابلیت حمل، قابلیت اصلاح و استفاده مجدد حمایت کنند
- راه‌حل:
  - برای دستیابی به جداسازی دغدغه‌ها الگوی لایه‌ای، نرم‌افزار را به واحدهایی تحت عنوان **لایه** تقسیم می‌کند. هر لایه گروهی از ماژول‌ها است که مجموعه‌ای منسجم از خدمات را ارائه می‌دهند

مثالی از الگوی لایه‌ای



لایه

هر لایه اجازه دارد تنها با لایه زیرین
خود ارتباط داشته باشد

راهکار الگوی لایه‌ای

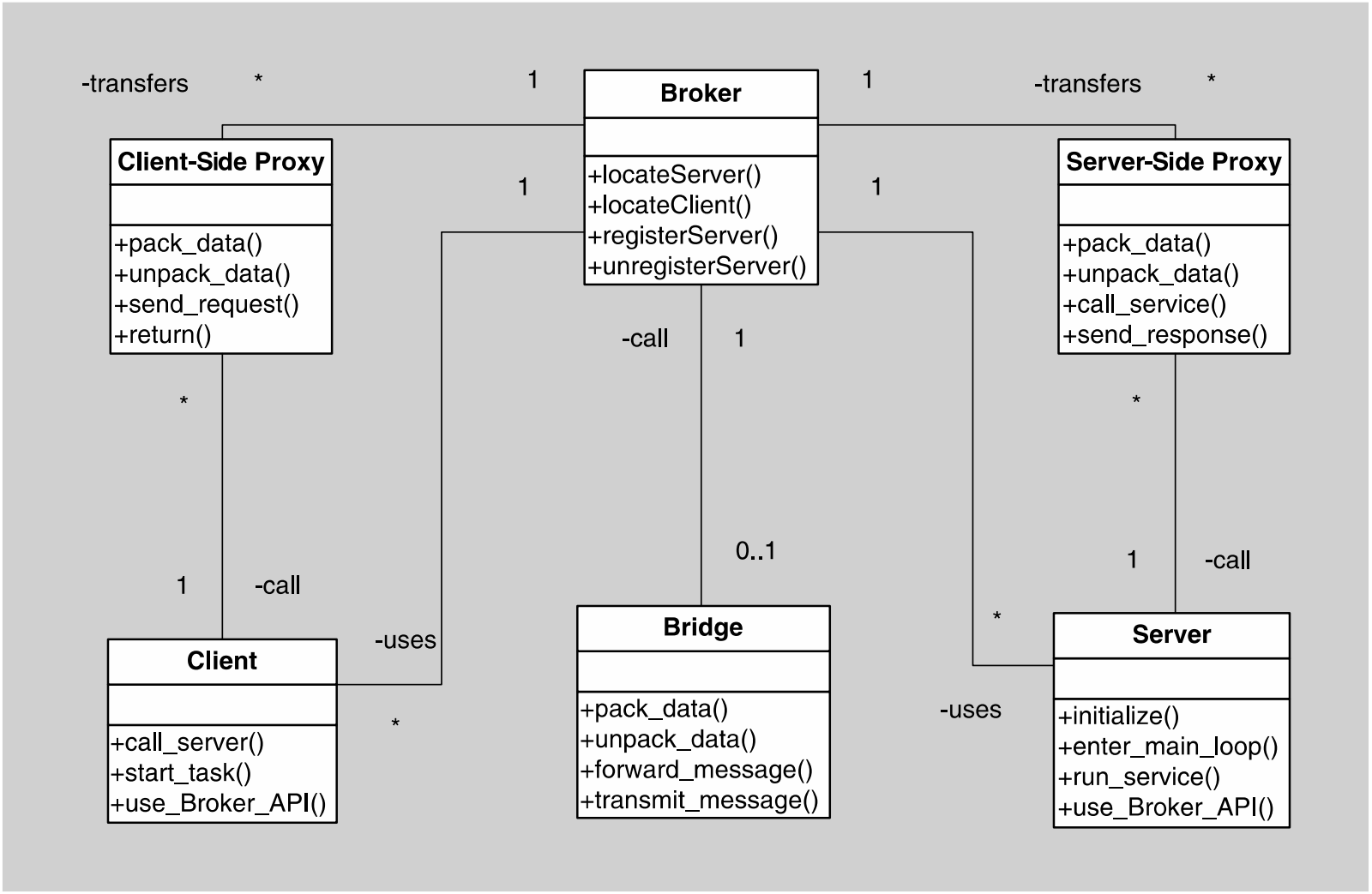
بررسی اجمالی	الگوی لایه‌ای، لایه‌ها (گروهی از ماژول‌ها که مجموعه‌ای از سرویس‌ها را ارائه می‌دهند) و ارتباط یک طرفه «اجازه استفاده دارد» را بین لایه‌ها تعریف می‌کند
عناصر	لایه، نوعی ماژول توصیف لایه باید بیانگر ماژول‌های تشکیل‌دهنده لایه و ویژگی‌های مجموعه منسجمی از خدمات که توسط لایه ارائه می‌شود، باشد
ارتباطات	اجازه استفاده دارد طراحی باید قواعد استفاده از لایه و استثنائات مجاز را مشخص کند (برای مثال: «یک لایه می‌تواند از هر لایه زیرین خود استفاده کند» یا «لایه مجاز است فقط از یک لایه زیرین خود استفاده نماید»)
محدودیت‌ها	هر بخش از نرم‌افزار دقیقا به یک لایه تخصیص داده می‌شود حداقل دو لایه وجود دارد (معمولا سه یا بیشتر لایه وجود دارد) رابطه‌های اجازه استفاده دارد نباید دایره‌وار باشند (لایه پایینی اجازه استفاده از لایه بالایی را ندارد)
نقاط ضعف	• افزودن لایه باعث افزایش هزینه و پیچیدگی سیستم می‌شود • لایه‌ها بر کارایی تأثیر منفی دارند

الگوهای مؤلفه و اتصال

الگوی Broker

- زمینه:
  - بیشتر سیستم‌ها از مجموعه‌ای از سرویس‌های توزیع‌شده بر روی سرورهای مختلف تشکیل شده‌اند. پیاده‌سازی چنین سیستم‌هایی پیچیده است چرا که دغدغه برای نحوه تعامل سیستم‌ها (نحوه اتصال آن‌ها به یکدیگر و مبادله اطلاعات) و قابلیت دسترسی به سرویس‌ها وجود دارد
- مسئله:
  - چگونه نرم‌افزار توزیع‌شده را ساختاردهی کنیم که کاربرانِ سرویس‌ها نیازی به اطلاع از ماهیت و مکان ارائه‌دهندگان سرویس نداشته باشند و در نتیجه تغییر پویای اتصال میان کاربران و ارائه‌دهندگان آسان شود؟
- راه‌حل:
  - الگوی ***Broker*** کاربران سرویس‌ها (سرویس‌گیرنده‌ها) را از ارائه‌دهندگان سرویس (سرویس‌دهنده‌ها) با قرار دادن یک واسط به نام ***broker*** جدا می‌کند. زمانی که سرویس‌گیرنده به سرویسی نیاز دارد، از طریق رابط سرویس، ***broker*** را جستجو می‌کند. سپس ***broker*** درخواست سرویس سرویس‌گیرنده را به سرویس‌دهنده انتقال می‌دهد

مثالی از الگوی Broker



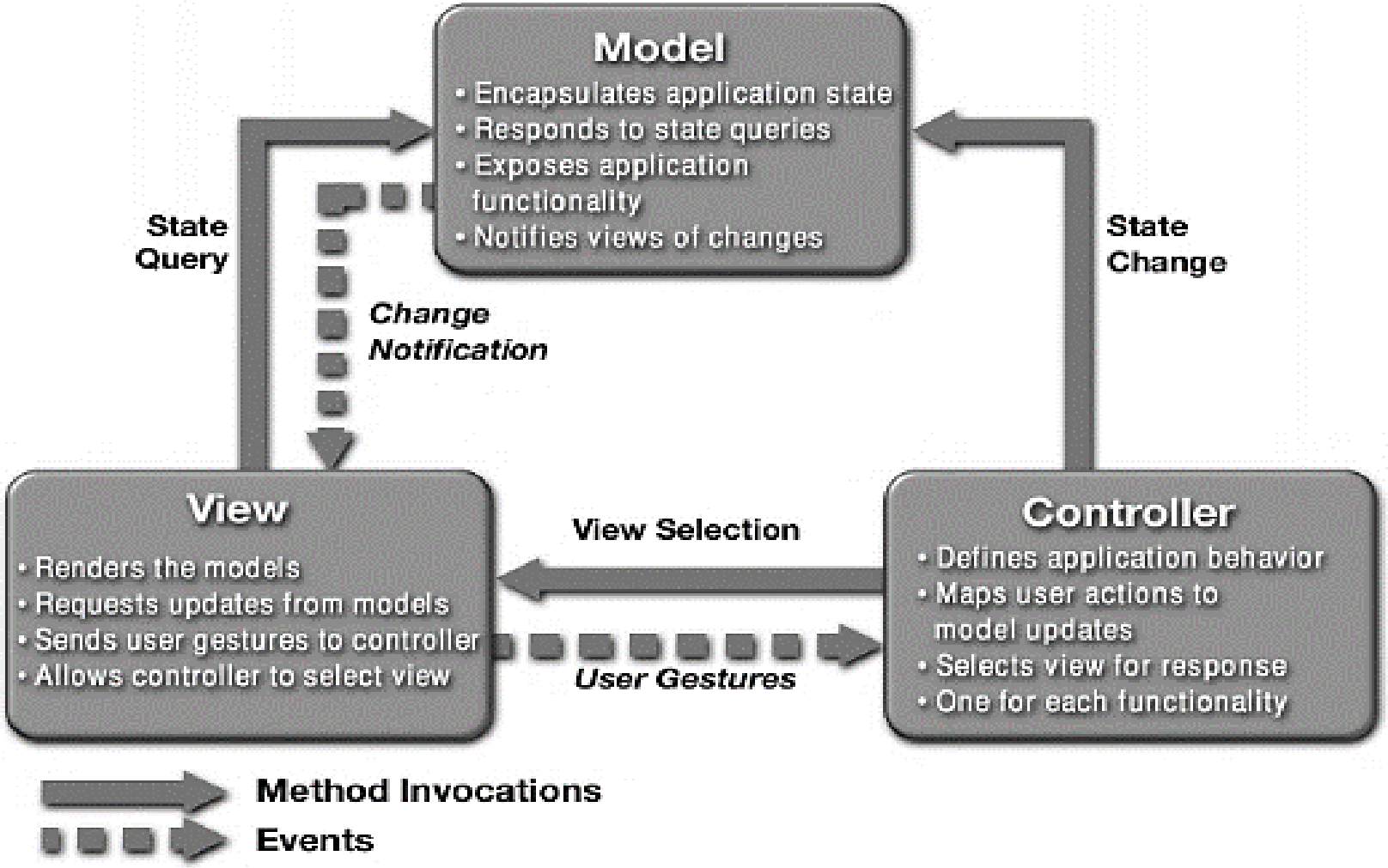
راهکار الگوی Broker

بررسی اجمالی	الگوی ***Broker*** مؤلفه زمان اجرایی به نام ***broker*** را تعریف می‌کند که واسط ارتباطی میان تعدادی از سرویس‌گیرندگان و سرویس‌دهندگان است
عناصر	سرویس‌گیرنده (***Client***): درخواست‌دهنده سرویس‌ها سرویس‌دهنده (***Server***): ارائه‌دهنده سرویس‌ها ***Broker***: واسطی که سرویس‌دهنده مناسب برای تأمین درخواست سرویس‌گیرنده را تعیین می‌کند، درخواست را به سرویس‌دهنده منتقل می‌کند و پاسخ را به سرویس‌گیرنده برمی‌گرداند پروکسی سمت سرویس‌گیرنده: واسطی که ارتباط با ***broker*** را مدیریت می‌کند پروکسی سمت سرویس‌دهنده: واسطی که ارتباط با ***broker*** را مدیریت می‌کند
ارتباطات	ارتباطات پیوست (***attachment***)، سرویس‌دهنده (و پروکسی سمت سرویس‌دهنده) و سرویس‌گیرنده (و پروکسی سمت گیرنده) را با ***broker*** مرتبط می‌سازند
محدودیت‌ها	سرویس‌گیرنده فقط می‌تواند به یک ***broker*** متصل باشد (به صورت بالقوه از طریق یک پروکسی سمت سرویس‌گیرنده) سرویس‌دهنده نیز تنها می‌تواند به یک ***broker*** متصل باشد (به صورت بالقوه از طریق یک پروکسی سمت سرویس‌دهنده)
نقاط ضعف	• ***broker*** لایه‌ای غیرمستقیم میان سرویس‌گیرنده‌ها و سرویس‌دهنده‌ها ایجاد می‌کند، این لایه می‌تواند تبدیل به گلوگاه ارتباطی تبدیل شود • ***broker*** می‌تواند نقطه شکست سیستم محسوب شود • ***broker*** می‌تواند سبب پیچیدگی شود • ***broker*** می‌تواند هدف حمله‌های امنیتی باشد • تست ***broker*** ممکن است مشکل باشد

الگوی MVC

- زمینه:
  - نرم‌افزار مربوط به رابط کاربری بخشی از نرم‌افزار تعاملی است که بیشترین اصلاح روی آن انجام می‌شود. کاربران اغلب مایلند که داده‌های خود را از منظرهای مختلف همچون نمودار میله‌ای یا نمودار دایره‌ای ببینند. نکته مهم این است که هردو نمایش از روی یک وضعیت یکسان از داده‌ها ایجاد می‌شود
- مسئله:
  - چگونه می‌توان کارکرد رابط کاربری نرم‌افزار را از سایر کارکردهای آن جدا کرد به طوری که همچنان نرم‌افزار قابلیت پاسخ به ورودی‌های کاربر و تغییرات اساسی داده‌های برنامه کاربردی را داشته باشد؟ و زمانی که این تغییرات اساسی رخ می‌دهد چگونه نمایش‌های مختلف از رابط کاربری ایجاد، نگهداری و هماهنگ می‌شوند؟
- راه‌حل:
  - الگوی ***MVC*** کارکرد نرم‌افزار را به سه نوع مؤلفه تقسیم می‌کند:
  - ***Model***: شامل داده‌های برنامه کاربردی است
  - ***View***: بخش‌هایی از داده‌های اصلی را نمایش می‌دهد و با کاربر در تعامل است
  - ***Controller*** : واسط بین ***Model*** و ***View*** است و اطلاع‌رسانی مربوط به تغییرات وضعیت را مدیریت می‌کند

مثالی از الگوی MVC



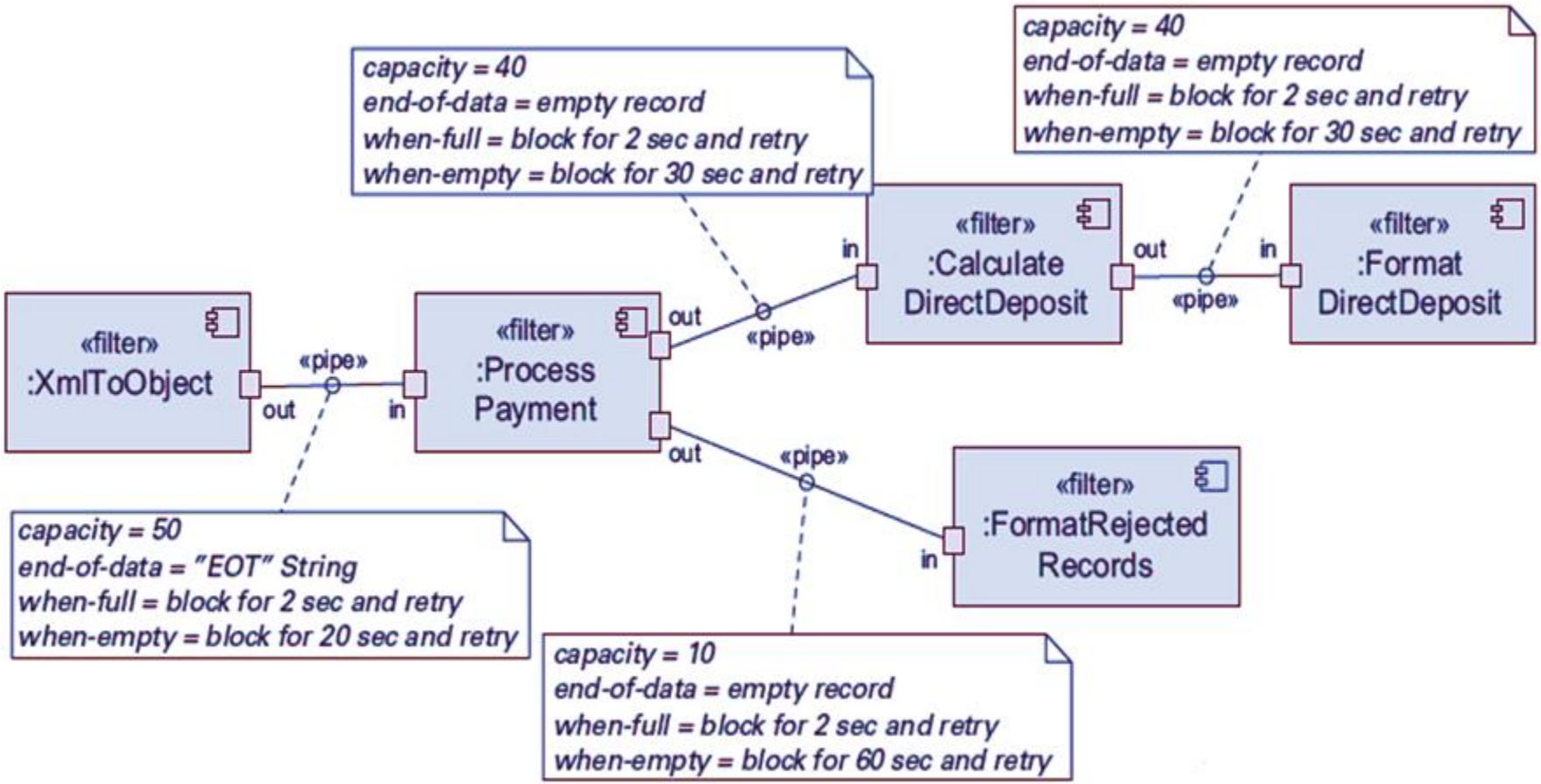
راهکار الگوی *MVC*

بررسی اجمالی	الگوی ***MVC*** کارکرد سیستم را به سه مؤلفه تقسیم می‌کند: ***model***، ***view*** و ***controller*** که واسطی میان ***model*** و ***view*** است
عناصر	***Model***: ارائه‌ای از داده‌های برنامه کاربردی یا وضعیت است و منطق برنامه کاربردی را شامل می‌شود (یا رابطی برای دستیابی به آن فراهم می‌کند) ***View***: مؤلفه رابط کاربر است که یا نمایشی از مدل را برای کاربر ایجاد می‌کند یا به نوعی اجازه ورود اطلاعات کاربر را می‌دهد ***Controller***: تعاملات میان ***model*** و ***view*** را مدیریت و اقدامات کاربر را به تغییرات ***model*** یا ***view*** تبدیل می‌کند
ارتباطات	ارتباط «اطلاع می‌دهد» (***notifies***) نمونه‌های ***model***، ***view*** و ***controller*** را به هم مرتبط می‌کند و تغییراتِ وضعیت را به عناصر اطلاع می‌دهد
محدودیت‌ها	از هر یک از مؤلفه‌های ***model***، ***view*** و ***controller*** باید حداقل یک نمونه ایجاد شود مؤلفه ***model*** نباید به طور مستقیم با مؤلفه ***controller*** در تعامل باشد
نقاط ضعف	شاید پیچیدگی ایجاد شده ارزش رابط کاربری ساده را نداشته باشد ممکن است تجریدهای ***model***، ***view*** و ***controller*** برای برخی ابزارهای رابط کاربری مناسب نباشند

الگوی *Pipe and Filter*

- زمینه:
  - بسیاری از سیستم‌ها باید جریان‌هایی از داده‌های گسسته را از ورودی به خروجی تبدیل کنند. بسیاری از انواع این تغییرات بارها در عمل رخ می‌دهد، پس تبدیل این تغییرات به بخش‌های مستقل و قابل استفاده مجدد مطلوب است
- مسئله:
  - این گونه سیستم‌ها باید به مؤلفه‌هایی با قابلیت استفاده مجدد و اتصال سست که از سازوکارهای ارتباطی ساده و عمومی بهره می‌برند، تقسیم شوند. بدین ترتیب امکان ترکیب انعطاف‌پذیر آن‌ها فراهم می‌شود. مؤلفه‌هایی که عمومی و با اتصال سست هستند به راحتی قابلیت استفاده مجدد دارند. مؤلفه‌هایی که مستقلند می‌توانند به صورت موازی اجرا شوند
- راه‌حل:
  - الگوی تعامل در الگوی ***Pipe and Filter*** با تغییرات پی در پی بر روی جریان‌های داده مشخص می‌شود. داده‌هایی که به پورت(های) ورودی یک فیلتر می‌رسند، تغییر کرده و سپس از طریق پورت(های) خروجی در طول ***Pipe*** عبور می‌کنند و به فیلتر بعدی وارد می‌شوند. در ضمن یک فیلتر می‌تواند به تنهایی داده‌ها را از یک یا چندین پورت مصرف کند یا داده برای آن پورت‌ها تولید کند

مثالی از الگوی *Pipe and Filter*



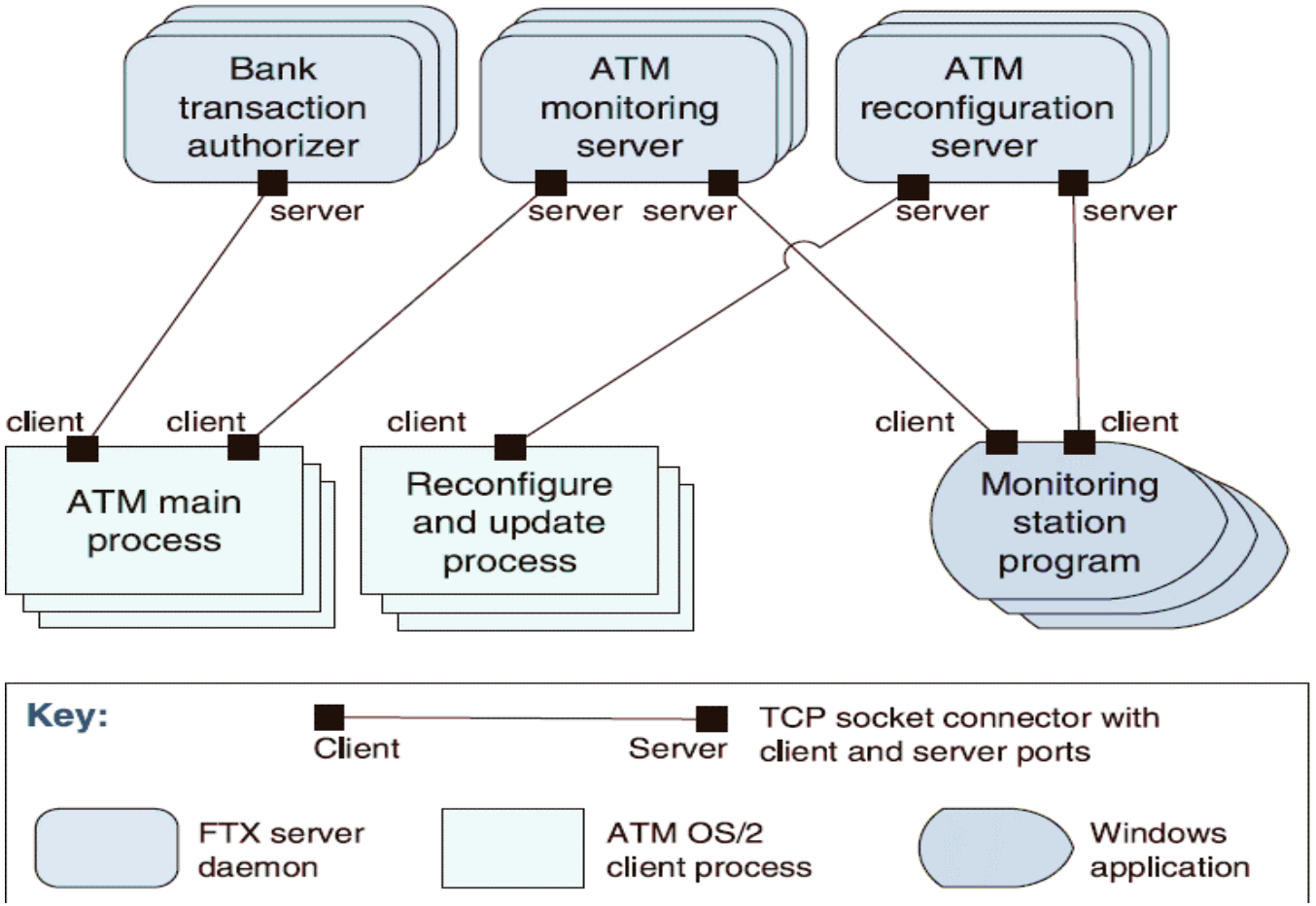
راهکار الگوی *Pipe and Filter*

بررسی اجمالی	داده از ورودی‌های سیستم پس از اعمال تغییرات پی در پی توسط فیلترهایی که به ***pipe*** متصل هستند، به خروجی‌های سیستم منتقل می‌شوند
عناصر	فیلتر: مؤلفه‌ای است که داده‌هایی را که از پورت(های) ورودی می‌خواند به داده‌هایی که بر روی پورت(های) خروجی نوشته می‌شوند، تبدیل می‌کند ***Pipe***: اتصال‌دهنده‌ای است که داده را از پورت(های) خروجی یک فیلتر به پورت(های) ورودی فیلتر دیگری انتقال می‌دهد. ***pipe*** دارای فقط یک منبع برای ورودی و یک هدف به عنوان خروجی‌اش است. ***pipe*** دنباله‌ای از اقلام داده را نگه می‌دارد و تغییری در داده‌های عبوری ایجاد نمی‌کند
ارتباطات	ارتباط پیوست (***attachment***)، خروجی فیلتر را به ورودی ***pipe*** مرتبط می‌کند و بالعکس
محدودیت‌ها	***Pipe*** پورت‌های خروجی فیلتر را به پورت‌های ورودی فیلتر متصل می‌کند فیلترها باید در مورد نوع داده‌ای که از طریق ***Pipe*** عبور می‌کند، باهم توافق داشته باشند
نقاط ضعف	الگوی ***Pipe and Filter*** گزینه مناسبی برای سیستم‌های تعاملی نیست وجود تعداد زیادی فیلتر مستقل می‌تواند سربار محاسباتی زیادی را ایجاد کند سیستم‌های ***Pipe and Filter*** برای محاسبات طولانی مدت مناسب نیست

الگوی سرویس‌گیرنده–سرویس‌دهنده (Client-Server)

- زمینه:
  - منابع و سرویس‌های مشترکی وجود دارند که تعداد زیادی از سرویس‌گیرندگان توزیع شده خواستار دسترسی به آن‌ها هستند و برای هر یک از آن‌ها باید دسترسی یا کیفیت سرویس کنترل شود
- مسئله:
  - با مدیریت مجموعه‌ای ازمنابع و سرویس‌ها، ما می‌توانیم قابلیت اصلاح و استفاده مجدد را با فاکتور گرفتن سرویس‌های مشترک و اصلاح آن‌ها در یک مکان بهبود ببخشیم. با وجود این که این منابع به صورت فیزیکی بر روی چندین سرویس‌دهنده توزیع شده‌اند، ما خواستار بهبود مقیاس‌پذیری و قابلیت دسترسی با متمرکز کردن کنترل این منابع و سرویس‌ها هستیم
- راه‌حل:
  - سرویس‌گیرنده‌ها با فراخوانی سرویس‌های سرویس‌دهنده تعامل برقرار می‌کنند. برخی مؤلفه‌ها ممکن است هم به عنوان سرویس‌گیرنده و هم سرویس‌دهنده عمل کنند. ممکن است یک سرویس‌دهنده مرکزی یا چندین سرویس‌دهنده توزیع‌شده داشته باشیم

مثالی از الگوی سرویس‌گیرنده–سرویس‌دهنده



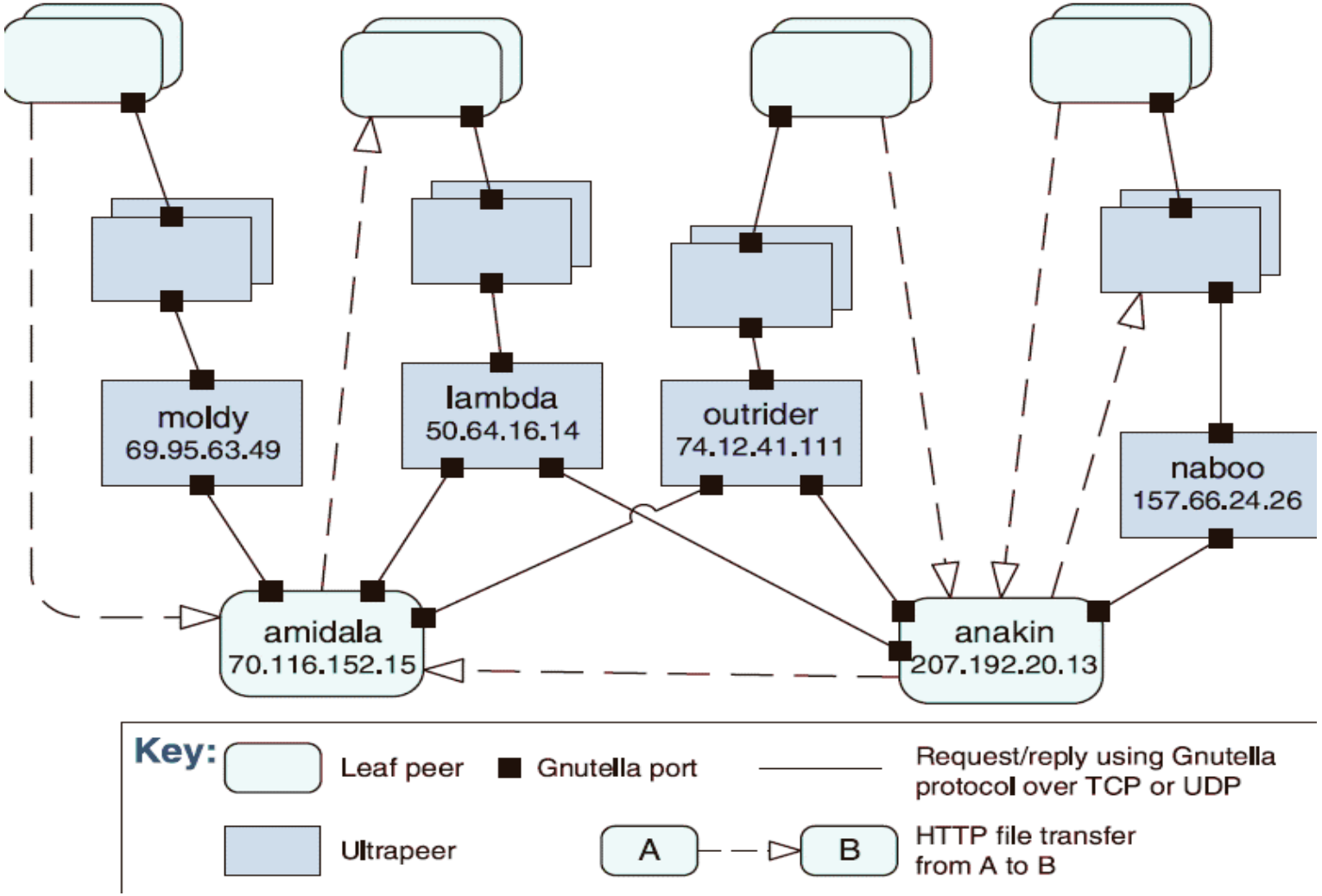
راهکار الگوی سرویس‌گیرنده–سرویس‌دهنده

بررسی اجمالی	سرویس‌گیرنده‌ها ارتباط با سرویس‌دهنده‌ها را آغاز می‌کند و به فراخوانی سرویس‌های مورد نیاز خود از سرورها پرداخته و منتظر پاسخ درخواست‌ها می‌ماند
عناصر	سرویس‌گیرنده (***client***): مؤلفه‌ای است که سرویس‌ها را از سرور فراخوانی می‌کند. سرویس‌گیرنده دارای پورت‌هایی هستند که سرویس‌های مورد نیاز آن‌ها را بیان می‌کنند سرویس‌دهنده (***server***): مؤلفه‌ای است که به ارائه سرویس می‌پردازد. سرورها نیز دارای پورت‌هایی هستند که سرویس‌های ارائه شده توسط آن‌ها را توصیف می‌کنند اتصال‌دهنده درخواست/پاسخ (***Request/reply connector***): یک اتصال داده‌ای است که پروتکل درخواست/پاسخ را به کار می‌گیرد و توسط سرویس‌گیرنده برای فراخوانی سرویس‌های سرور استفاده می‌شود. محلی یا از راه دور بودن فراخوانی یا این که داده رمزگذاری شده یا نشده است از ویژگی‌های مهم محسوب می‌شوند
ارتباطات	ارتباط پیوست، سرویس‌گیرنده را به سرویس‌دهنده مرتبط می‌سازد
محدودیت‌ها	سرویس‌گیرنده‌ها از طریق اتصال‌دهنده درخواست/پاسخ به سرورها متصل هستند مؤلفه‌های سرویس‌دهنده می‌توانند خود از سرورهای دیگر سرویس بگیرند
نقاط ضعف	سرور می‌تواند گلوگاه کارایی باشند سرور می‌تواند نقطه شکست باشد تصمیمات در خصوص محل قرارگیری کارکرد (در سرویس‌گیرنده یا سرویس‌دهنده) پیچیده است و هزینه تغییر آن پس از ساخت سیستم زیاد خواهد بود

الگوی Peer-to-Peer

- زمینه:
  - موجودیت‌های محاسباتی توزیع‌شده (که همه آن‌ها از لحاظ آغاز یک تعامل از اهمیت یکسانی برخوردارند و هر یک منابع خود را تأمین می‌کند) باید برای ارائه سرویس به مجموعه‌ای از کاربران که آن‌ها نیز توزیع‌شده‌اند با یکدیگر همکاری کنند
- مسئله:
  - چگونه مجموعه‌ای از این موجودیت‌های محاسباتی توزیع‌شده می‌توانند به واسطه یک پروتکل مشترک به یکدیگر متصل شوند و سرویس‌های خود را با قابلیت دسترسی و مقیاس‌پذیری بالا مدیریت کرده و به اشتراک بگذارند
- راه‌حل:
  - در الگوی $P2P$ مؤلفه‌ها مستقیما با هم به عنوان عضو ($peer$) تعامل برقرار می‌کنند. تمام اعضا با هم برابر هستند. ارتباطات $P2P$ معمولا به صورت تعاملات درخواست/پاسخ است.

مثالی از الگوی *Peer-to-Peer*



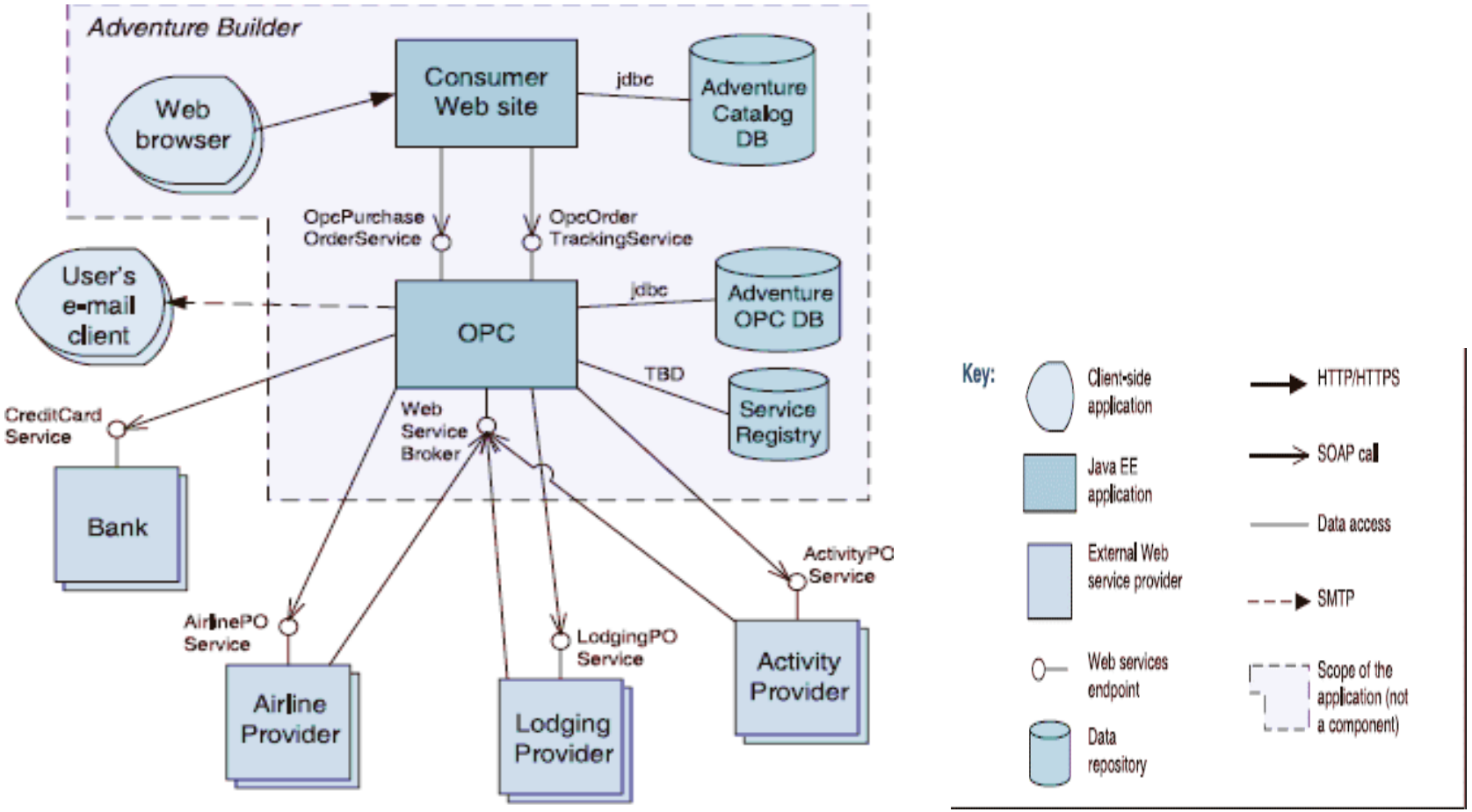
راهکار الگوی *Peer-to-Peer*

بررسی اجمالی	محاسبات با همکاری تعدادی عضو که از یکدیگر درخواست سرویس‌کنند و به یکدیگر سرویس ارائه می‌کنند، انجام می‌شود
عناصر	عضو (***peer***): مؤلفه مستقلی است که روی یک نود شبکه اجرایی می‌شود. برخی از مؤلفه‌های عضو قابلیت مسیریابی، نمایه‌سازی و جستجوی دیگر اعضا را دارند اتصال‌دهنده درخواست/پاسخ: برای اتصال به شبکه، جستجوی دیگر اعضا و فراخوانی سرویس‌ها از دیگر اعضا استفاده می‌شود
ارتباطات	ارتباط، اعضا را با یکدیگر مرتبط می‌کند. در زمان اجرا امکان تغییر پیوست‌ها (***attachments***) وجود دارد
محدودیت‌ها	محدودیت‌ها ممکن است روی هر یک از موارد زیر اعمال شوند: • تعداد پیوست مجاز برای هر عضو • تعداد ***hop*** مورد استفاده برای یافتن یک عضو • کدام عضو در مورد عضو دیگر اطلاع دارد برخی از شبکه‌های ***P2P*** با استفاده از توپولوژی ***Star*** سازماندهی شده‌اند که در آن هر عضو تنها به ***supernodes*** متصل است
نقاط ضعف	مدیریت امنیت، سازگاری داده‌ها، دسترسی داده/سرویس، پشتیبان‌گیری و بازیابی پیچیده است سیستم‌های ***P2P*** کوچک ممکن است قادر به دستیابی متداوم به اهداف کیفی مانند کارایی و قابلیت دسترسی نباشد

الگوی معماری سرویس‌گرا

- زمینه:
  - تعدادی سرویس توسط ارائه‌دهندگان سرویس توصیف و ارائه می‌شود و توسط مصرف‌کننده سرویس استفاده می‌شود. مصرف‌کنندگان سرویس باید قادر به درک و استفاده از این سرویس‌ها بدون نیاز به اطلاع از جزئیات پیاده‌سازی آن‌ها باشند
- مسئله:
  - چگونه می‌توان میان مؤلفه‌های توزیع‌شده که بر روی سکوهای مختلف اجرا می‌شوند، با زبان‌های مختلف نوشته شده‌اند، توسط سازمان‌های مختلف ارائه می‌شوند و در سراسر اینترنت توزیع شده‌اند تعامل‌پذیری (***Interoperability***) ایجاد کنیم؟
- راه‌حل:
  - الگوی معماری سرویس‌گرا (***SOA***) مجموعه‌ای است از مؤلفه‌های توزیع‌شده که ارائه‌دهنده یا مصرف‌کننده سرویس هستند

مثالی از الگوی معماری سرویس‌گرا



راهکار الگوی معماری سرویس‌گرا

بررسی اجمالی	محاسبات با کمک مجموعه‌ای از مؤلفه‌های همکار که در یک شبکه به ارائه یا مصرف سرویس می‌پردازند، انجام می‌شود
عناصر	مؤلفه‌ها: • ارائه‌دهندگان سرویس: یک یا چندین سرویس را از طریق واسط‌های منتشر شده ارائه می‌دهد • مصرف‌کنندگان سرویس: سرویس‌ها را به صورت مستقیم یا از طریق یک واسط فراخوانی می‌کنند • ارائه‌دهندگان سرویس خود می‌توانند مصرف‌کننده سرویس نیز باشند گذرگاه سرویس سازمانی (*ESB*): واسطی است که می‌تواند به مسیریابی و تبدیل پیام‌های رد و بدل شده میان ارائه‌دهنده و مصرف‌کننده سرویس بپردازد رجیستری سرویس‌ها: ارائه‌دهنده‌ها برای ثبت سرویس‌ها و مصرف‌کنندگان برای کشف سرویس‌ها در زمان اجرا از آن استفاده می‌کنند سرور همنواسازی (*Orchestration Server*): تعاملات میان مصرف‌کنندگان و ارائه‌دهندگان سرویس را بر اساس زبان‌های فرایندها و جریان‌کاری حرفه هماهنگ می‌کند اتصال‌دهنده‌ها: • *SOAP connector*: از پروتکل *SOAP* برای ارتباطات همگام میان سرویس‌های وب، که معمولا از طریق *HTTP* برقرار می‌شود، استفاده می‌کند • *REST connector*: متکی به عملیات ابتدایی درخواست/پاسخ در پروتکل *HTTP* است • *Asynchronous messaging connector*: از یک سیستم پیام‌رسانی برای تبادل ناهمگن پیام به صورت *point-to-point* یا *publish-subscribe* استفاده می‌کند

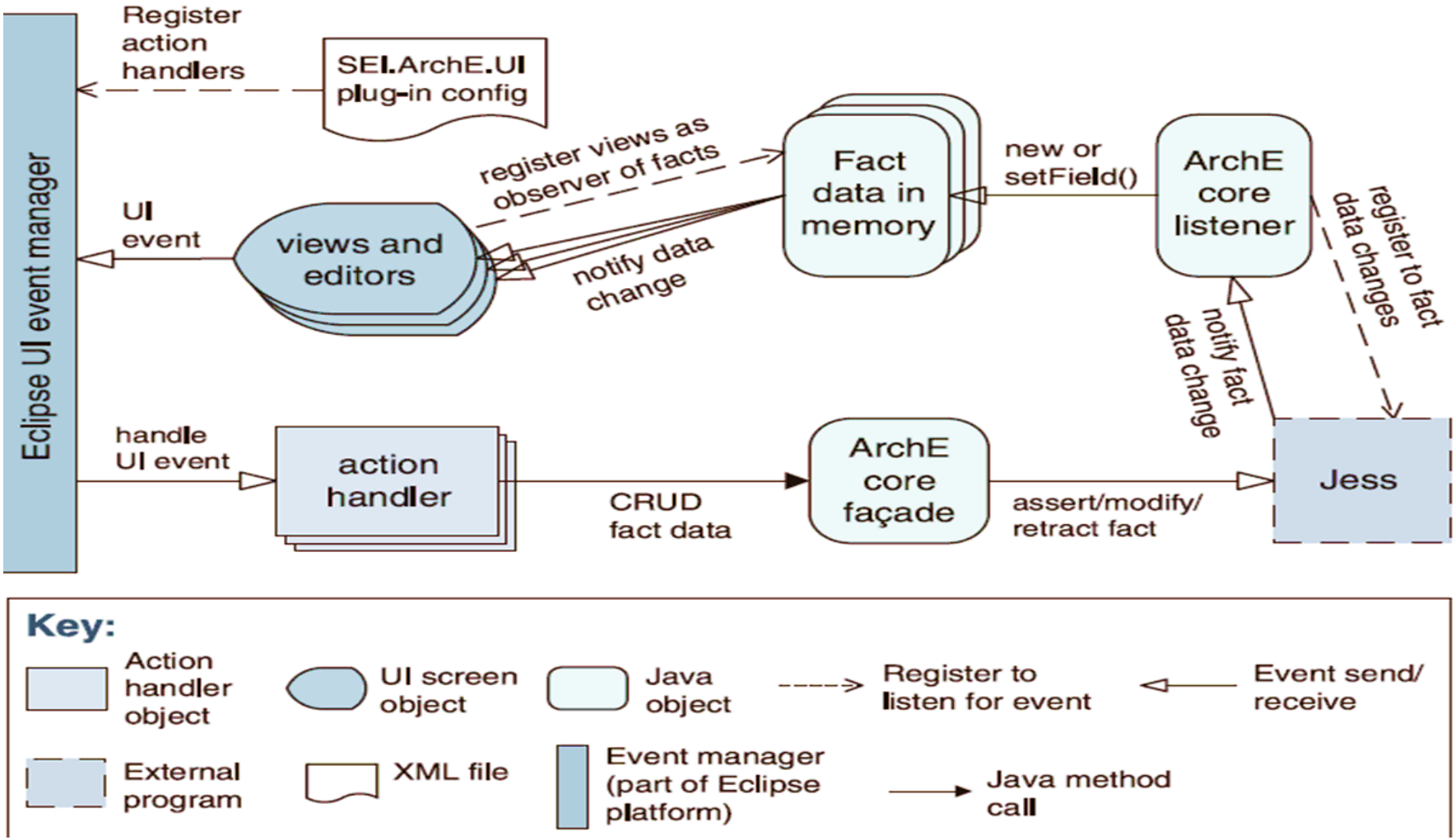
راهکار الگوی معماری سرویس‌گرا (ادامه)

ارتباطات	پیوستِ انواع مختلفی از مؤلفه‌های در دسترس به اتصالات مربوط
محدودیت‌ها	مصرف‌کنندگان سرویس به ارائه‌دهندگان متصل هستند، اما ممکن است در این اتصال از مؤلفه‌های واسط (مانند ***ESB***، رجیستری و سرور همنواسازی) استفاده شود
نقاط ضعف	ساخت سیستم‌های مبتنی بر معماری سرویس‌گرا معمولا پیچیده است شما کنترلی بر تکامل سرویس‌های مستقل ندارید سربار کارایی به دلیل وجود میان‌افزار (***middleware***) وجود دارد. همچنین سرویس‌ها ممکن است گلوگاه کارایی باشند و معمولا تضمینی برای کارایی ارائه نمی‌دهند

الگوی Publish-Subscribe

- **زمینه:**
  - تعدادی ارائه‌دهنده و مصرف‌کننده داده که باید با هم تعامل داشته باشند، وجود دارد. نه تعداد دقیق و ماهیت ارائه‌دهندگان و مصرف‌کنندگان داده از پیش تعیین شده یا ثابت است و نه داده‌هایی که آن‌ها به اشتراک می‌گذارند
- **مسئله:**
  - چگونه می‌توان سازوکارهای یکپارچه‌سازی را برای حمایت از انتقال پیام‌ها بین ارائه‌دهندگان و مصرف‌کنندگان ایجاد کرد به گونه‌ای که آن‌ها از هویت یکدیگر یا حتی وجود هم اطلاع نداشته باشند؟
- **راه‌حل:**
  - در الگوی ***publish-Subscribe***، مؤلفه‌ها از طریق پیام‌ها یا رخدادها با هم در تعامل هستند. ممکن است مؤلفه‌ها در دریافت یک سری از رخدادها با یکدیگر مشترک باشند. مؤلفه‌های ***publisher*** رخدادها را به وسیله قرار دادن آن‌ها بر روی گذرگاه به اطلاع می‌رسانند؛ سپس اتصال دهنده‌ها رخدادها را به مؤلفه‌های ***subscribe*** که خواهان این رخدادها بودند، می‌رسانند

مثالی از الگوی *Publish-Subscribe*



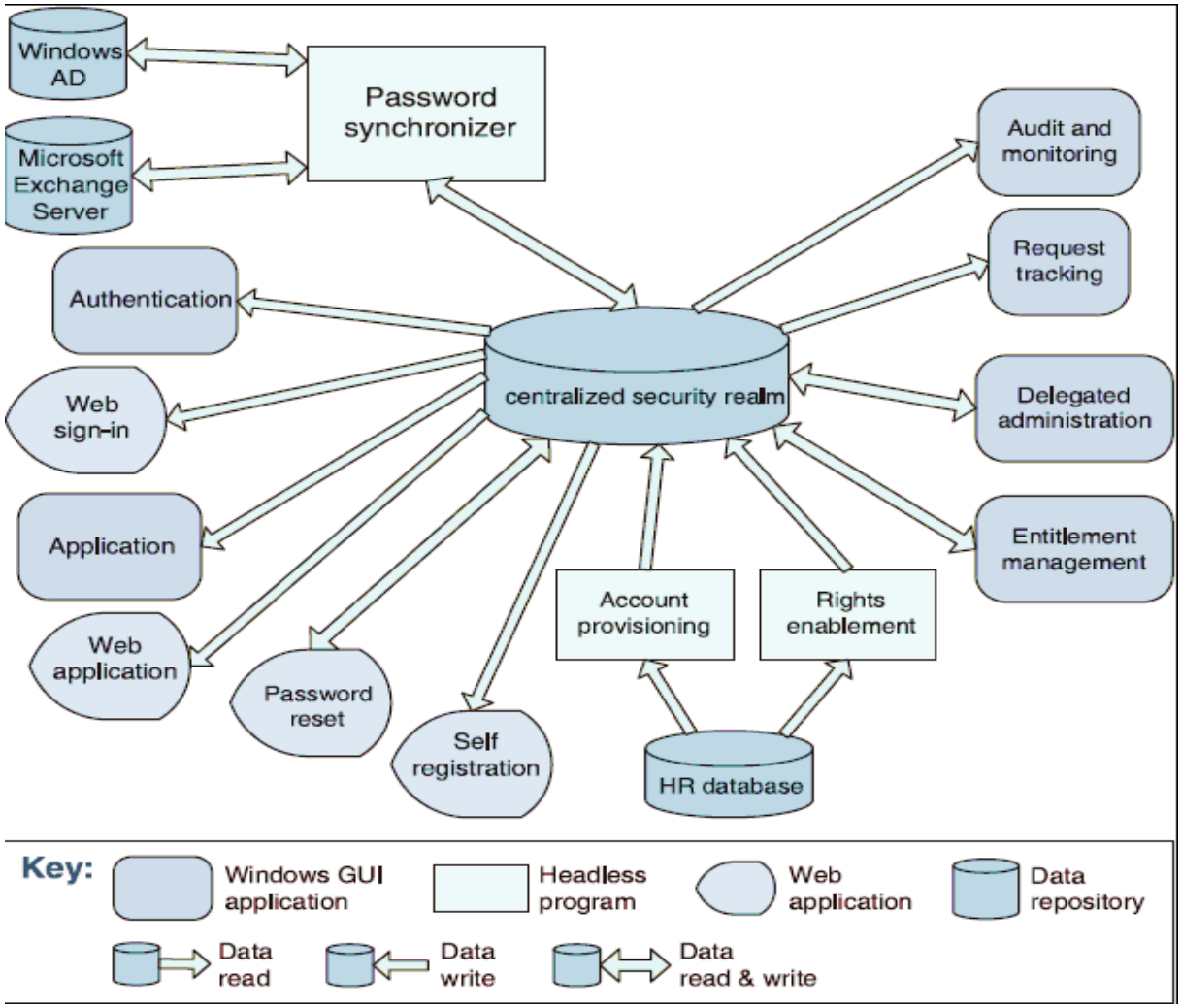
راهکار الگوی Publish-Subscribe

بررسی اجمالی	مؤلفه‌ها به انتشار رخدادها و اشتراک برای دریافت رخدادها می‌پردازند. زمانی که یک رخداد به وسیله یک مؤلفه اعلام گردید، زیرساخت اتصال، رخداد را برای تمام مؤلفه‌هایی که خواهان این رخداد بودند می‌فرستد
عناصر	هر مؤلفه ***C&C*** که حداقل یک پورت ***publish*** یا یک پورت ***subscribe*** داشته باشد اتصال ***publish-subscribe***: به اعلام رخداد یا گوش دادن به رخدادها برای مؤلفه‌هایی که تمایل به انتشار یا دریافت رخدادها دارند، می‌پردازد
ارتباطات	ارتباط پیوست، مؤلفه‌ها را به اتصال‌دهنده ***publish-subscribe*** مرتبط می‌کند. این ارتباط تعیین می‌کند که کدام یک از مؤلفه‌ها به اعلام رخدادها می‌پردازند و کدام یک خواهان دریافت رخدادها هستند
محدودیت‌ها	تمام مؤلفه‌ها به یک توزیع‌کننده رخداد متصل هستند. این توزیع‌کننده رخداد می‌تواند یک گذرگاه (اتصال) یا یک مؤلفه باشد. پورت‌های ***publish*** به نقش‌های مربوط به اعلام و پورت‌های ***subscribe*** به نقش‌های مربوط به گوش دادن (***listen roles***) متعلق هستند
نقاط ضعف	به طور معمول تأخیر را افزایش می‌دهد و بر مقیاس‌پذیری و قابلیت پیش‌بینی زمان رسیدن پیام‌ها تأثیر منفی دارد کنترل کمتر بر ترتیب پیام‌ها و عدم تضمین رسیدن پیام‌ها

الگوی داده مشترک (*Shared-Data*)

- **زمینه:**
  - مؤلفه‌های محاسباتی مختلف باید به اشتراک‌گذاری و تغییر تعداد زیادی از داده‌ها بپردازند. این داده‌ها فقط به یکی از مؤلفه‌ها تعلق ندارد
- **مسئله:**
  - سیستم‌ها چگونه می‌توانند داده‌های پایا را که در دسترس مؤلفه‌های مستقل هستند ذخیره کنند و تغییر دهند؟
- **راه‌حل:**
  - در الگوی داده مشترک، تعاملات به وسیله تبادل داده پایا میان چندین ***data accessor*** و حداقل یک مخزن داده مشترک صورت می‌پذیرد. تبادل ممکن است توسط ***data accessor*** یا مخزن داده آغاز شود

مثالی از الگوی داده مشترک



راهکار الگوی داده مشترک

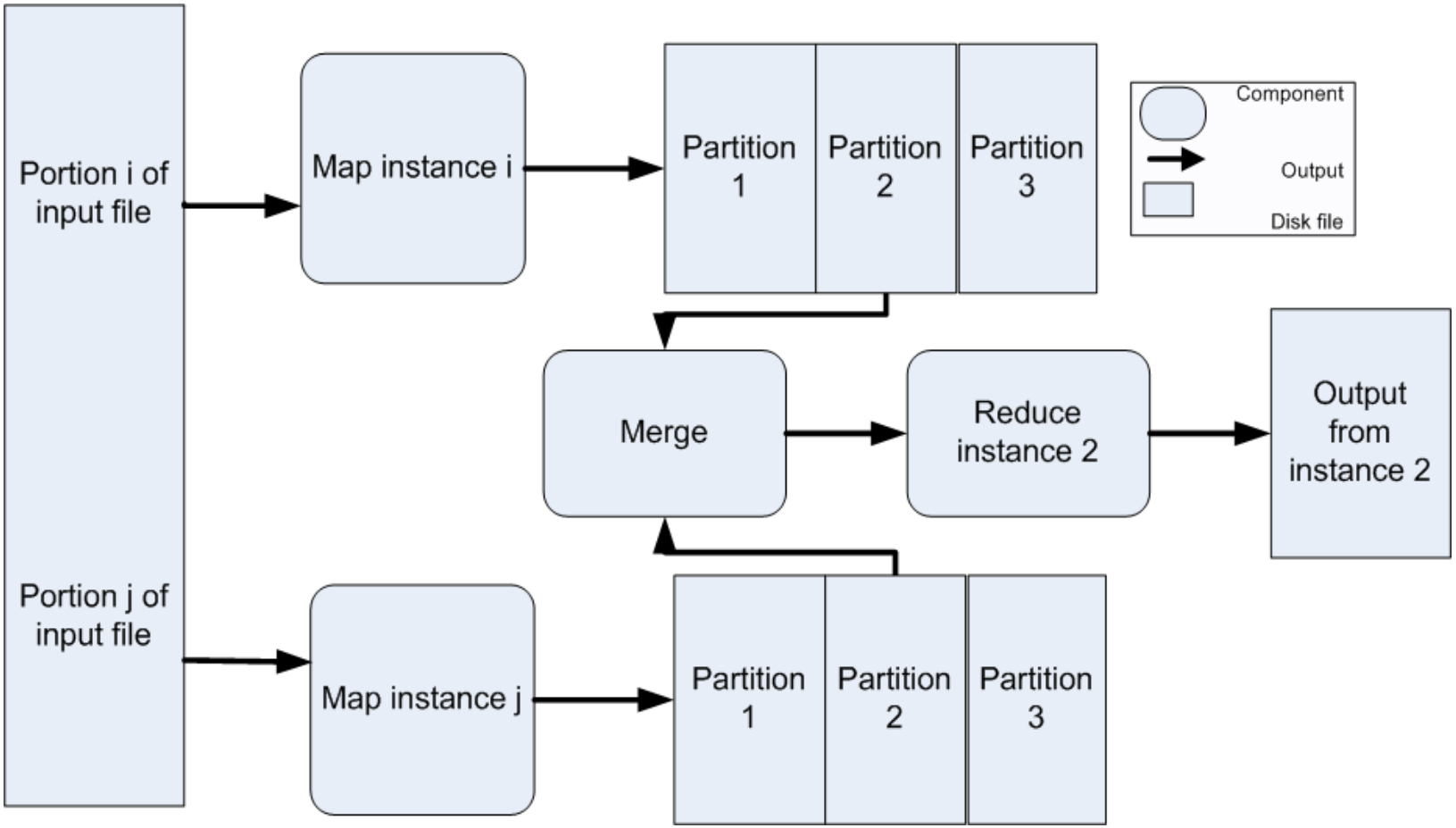
بررسی اجمالی	ارتباط میان ***data accessor*** توسط یک مخزن داده مشترک صورت می‌پذیرد. کنترل ممکن است توسط ***data accessors*** یا مخزن داده آغاز شود. داده با ذخیره‌سازی در مخزن داده پایا می‌شود
عناصر	مخزن داده مشترک: دغدغه‌های مربوط به آن شامل انواع داده ذخیره شده، ویژگی‌های مرتبط با کارایی داده، توزیع‌شدگی داده و تعداد ***accessor*** مجاز مؤلفه ***data accessor*** اتصال خواندن و نوشتن داده
ارتباطات	ارتباط پیوست مشخص می‌کند کدام ***data accessor*** به کدام مخزن داده متصل است
محدودیت‌ها	***data accessor*** تنها با مخزن(های) داده در تعامل هستند
نقاط ضعف	مخزن داده مشترک ممکن است گلوگاه کارایی شود مخزن داده مشترک ممکن است تبدیل به نقطه شکست شود تولیدکنندگان و مصرف‌کنندگان داده ممکن است به هم اتصال تنگاتنگی داشته باشند

الگوهای تخصیص

الگوی Map-Reduce

- زمینه:
  - حرفه نیاز مبرمی به تجزیه و تحلیل سریع حجم زیادی از داده‌های تولید شده یا در دسترس دارد که این داده‌ها در مقیاس پتابایت هستند
- مسئله:
  - برای بسیاری از برنامه‌های کاربردی با داده‌های بسیار زیاد، طبقه‌بندی داده‌ها و سپس تحلیل داده‌های گروه‌بندی شده کافی است. طبقه‌بندی کارای داده‌های بسیار زیاد به صورت توزیع‌شده و موازی و ارائه راهکاری ساده برای برنامه‌نویس برای تعیین نوع تحلیل‌ها، مسئله‌ای است که الگوی ***Map-Reduce*** به حل آن می‌پردازد
- راه‌حل:
  - الگوی ***Map-Reduce*** نیازمند سه بخش است:
  - یک زیر ساخت تخصصی که مراقب تخصیص نرم‌افزار به نودهای سخت‌افزاری در محیط محاسباتی پردازش موازی باشد و طبقه‌بندی داده‌ها را مدیریت کند
  - یک مؤلفه تعریف شده توسط برنامه‌نویس که به آن ***map*** می‌گویند که داده‌ها را به منظور بازیابی اقلامی که با هم ترکیب شده‌اند، فیلتر می‌کند
  - یک مؤلفه تعریف شده توسط برنامه‌نویس که به آن ***reduce*** می‌گویند که نتایج ***map*** را ادغام می‌کند

مثالی از الگوی *Map-Reduce*



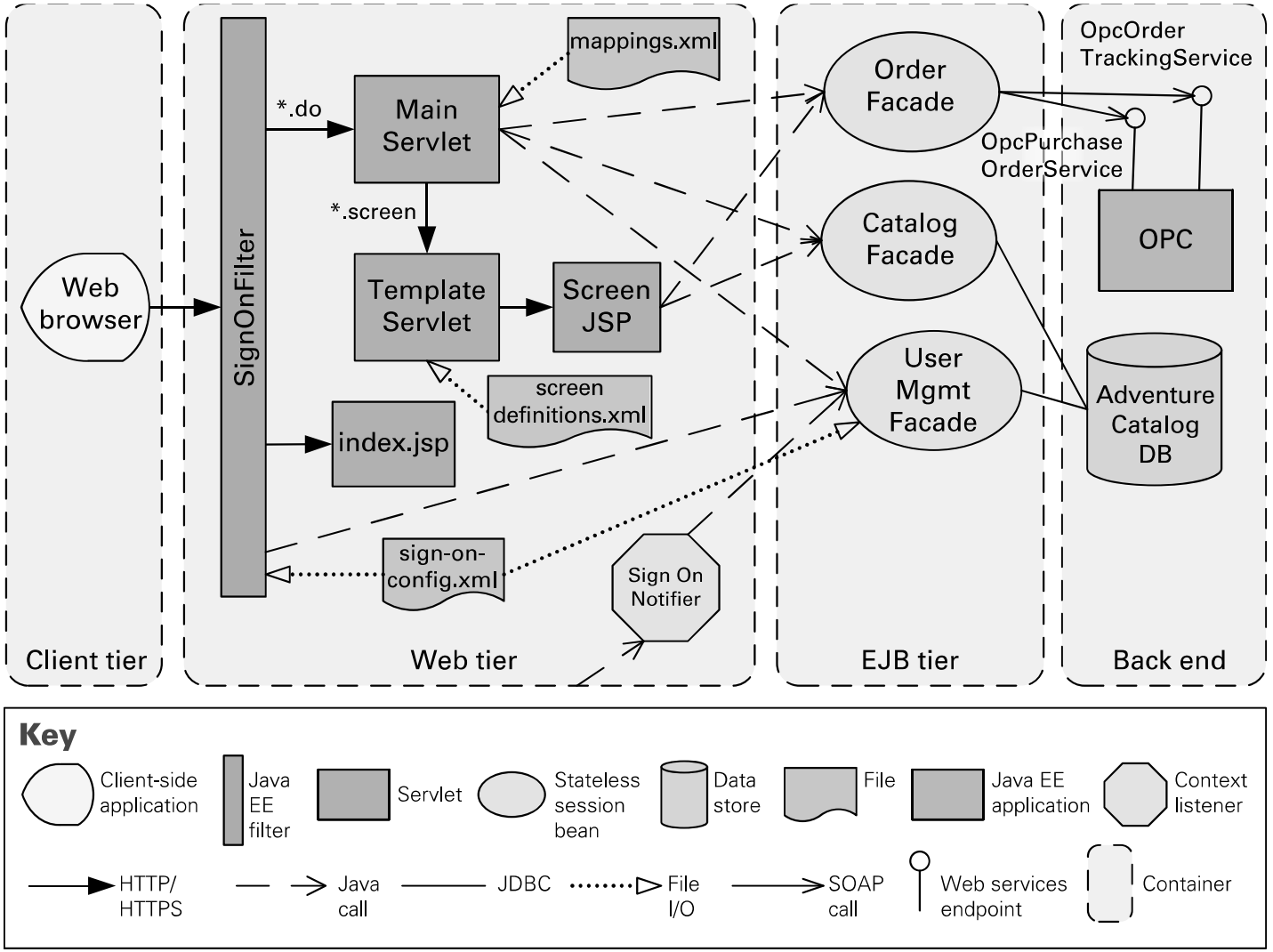
راهکار الگوی *Map-Reduce*

بررسی اجمالی	الگوی ***map-reduce*** چارچوبی را برای تحلیل مجموعه بزرگی از داده‌های توزیع‌شده که به صورت موازی بر روی مجموعه‌ای از پردازنده‌ها اجرا می‌شود، فراهم می‌آورد. این موازی‌سازی تأخیر کم و دسترسی بالا را به ارمغان می‌آورد. ***map*** وظیفه استخراج و تبدیل بخش‌هایی از تحلیل را بر عهده دارد و ***reduce*** بارگذاری نتیجه را اجرا می‌کند
عناصر	***Map***: یک تابع با نمونه‌های مختلف است که در چندین پردازنده استقرار یافته و وظیفه استخراج و تبدیل بخش‌هایی از تحلیل را بر عهده دارد ***Reduce***: تابعی است که می‌تواند به عنوان یک نمونه واحد یا چند نمونه در پردازنده‌ها برای اجرای بخشی از عملیات ***Extract-Transform-Load(ETL)*** استقرار یابد زیرساخت: چارچوبی که مسئول استقرار نمونه‌های ***map*** و ***reduce***، هدایت داده میان آن‌ها و تشخیص شکست و بازیابی از آن است
ارتباطات	استقرار می‌یابد: رابطه‌ای میان یک نمونه از ***map*** یا ***reduce*** و پردازشگری است که روی آن نصب می‌شود نمونه‌سازی، پایش و کنترل: رابطه‌ای میان زیرساخت و نمونه‌های ***map*** و ***reduce*** است
محدودیت‌ها	داده‌های مورد تحلیل باید به صورت مجموعه‌ای از فایل‌ها موجود باشند توابع ***map*** بدون وضعیت (***stateless***) هستند و با یکدیگر ارتباط ندارند تنها ارتباط میان نمونه‌های ***map*** و ***reduce*** داده‌ای در قالب <کلید،مقدار> است که از نمونه‌های ***map*** ارسال می‌شود
نقاط ضعف	اگر حجم داده زیاد نیست، سربار الگوی ***map-reduce*** قابل توجیه نخواهد بود اگر مجموعه‌های داده را نمی‌توان به زیر مجموعه‌هایی با اندازه مشابه تقسیم کرد، مزایای موازی‌سازی از دست خواهد رفت همنواسازی عملیاتی که نیازمند چندین ***reduce*** هستند، پیچیده است

الگوی Multi-Tier

- زمینه:
  - در استقرار توزیع‌شده، اغلب نیاز به توزیع زیرساخت‌های سیستم به زیر مجموعه‌های مجزا وجود دارد
- مسئله:
  - چگونه می‌توان سیستم را به تعدادی ساختار اجرایی مستقل از لحاظ محاسباتی (گروه‌هایی از نرم‌افزار و سخت‌افزار) که از طریق رسانه‌های ارتباطی با هم تعامل دارند تقسیم نمود؟
- راه‌حل:
  - ساختار اجرایی بسیاری از سیستم‌ها در قالب مجموعه‌ای از گروه‌بندی‌های منطقی مؤلفه‌ها سازماندهی می‌شود. به هر گروه‌بندی یک ***tier*** گفته می‌شود

مثالی از الگوی Multi-Tier



راهکار الگوی *Multi-Tier*

بررسی اجمالی	ساختار اجرایی بسیاری از سیستم‌ها در قالب مجموعه‌ای از گروه‌بندی منطقی مؤلفه‌ها سازماندهی می‌شود. به هر گروه‌بندی یک ***tier*** گفته می‌شود
عناصر	***Tier***: گروه‌بندی منطقی مؤلفه‌های نرم‌افزاری
ارتباطات	بخشی است از: برای گروه‌بندی مؤلفه‌ها در ***tier*** ارتباط برقرار می‌کند با: برای نمایش نحوه تعامل ***tiers*** و مؤلفه‌های تشکیل‌دهنده آن‌ها با یکدیگر تخصیص داده شده به: در صورتی که ***tier*** به سکوهای محاسباتی نگاشت شوند
محدودیت‌ها	یک مؤلفه نرم‌افزاری دقیقا متعلق به یک ***tier*** است
نقاط ضعف	هزینه و پیچیدگی زیاد

ارتباط الگو و تاکتیک

- تاکتیک‌ها بلوک‌های سازنده معماری هستند که الگوهای معماری از آن‌ها ساخته می‌شوند
  - تاکتیک‌ها مانند اتم و الگوهای همانند مولکول هستند
- برای مثال الگوی لایه‌ای را می‌توان ترکیبی از تاکتیک‌ها زیر در نظر گرفت:
  - افزایش انسجام معنایی (*increase semantic coherence*)
  - سرویس‌های انتزاعی مشترک (*abstract common services*)
  - *Encapsulate*
  - محدود کردن مسیرهای ارتباطی (*restrict communication paths*)
  - استفاده از واسط (*use an intermediary*)

استفاده از تاکتیک‌ها برای بهبود الگوها

- تاکتیک‌ها یک مشکل خاص را بر طرف می‌کنند اما در برابر دیگر ویژگی‌های کیفی بی‌اثر یا دچار نقطه ضعف هستند
- الگوی ***Broker*** را در نظر بگیرید:
  - می‌تواند دارای گلوگاه‌های کارایی باشد
  - می‌تواند دارای یک نقطه شکست باشد
- بهبود الگوی ***Broker*** با کمک تاکتیک‌ها:
  - افزایش منابع می‌تواند به کارایی کمک کند
  - نگهداری نسخه‌های متعدد به قابلیت دسترسی کمک می‌کند

استفاده از تاکتیک‌ها در کنار هم

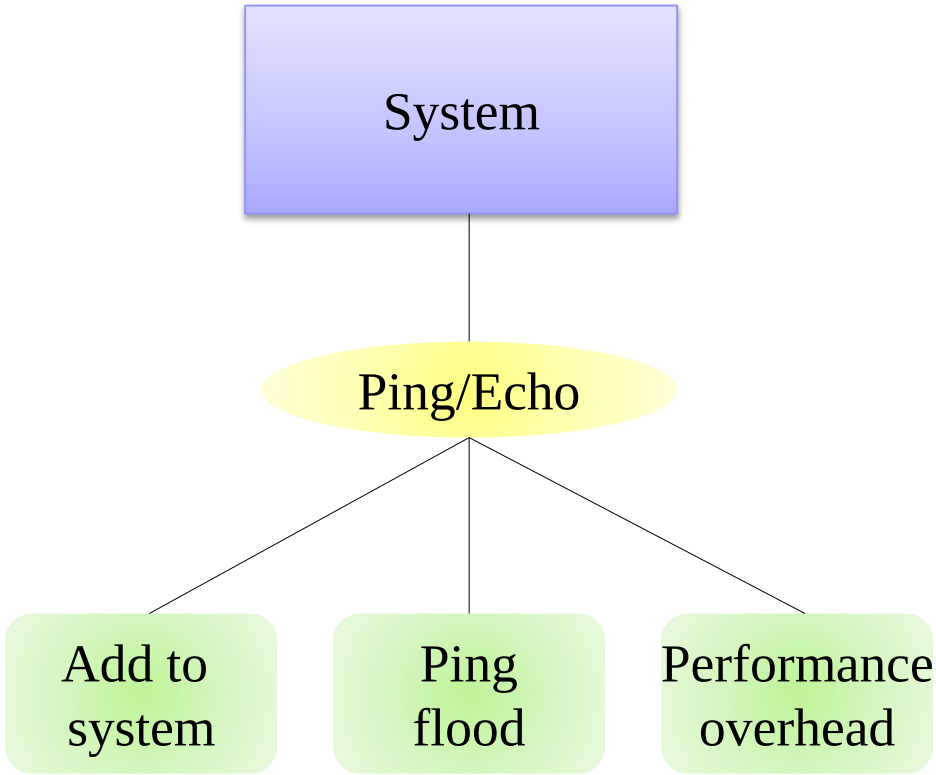
- هر تاکتیک دارای یکسری اثرات اصلی (برای مدیریت کارایی، قابلیت اصلاح و ...) و اثرات جانبی است
- در واقع در استفاده از هر تاکتیک باید مصالحه کرد
- ما قادر به استفاده سودآورد از تاکتیک‌ها هستیم چرا که می‌توان اثرات مستقیم و جانبی یک تاکتیک را اندازه‌گیری کرد و زمانی که مصالحه قابل قبول بود، تاکتیک را اعمال نمود
  - دستیابی به منفعت در ویژگی کیفی مورد نظر در حالی که چیز دیگری را از دست می‌دهیم

مثالی از تاکتیک‌ها و تعاملات

- سیستمی را در نظر بگیرید که باید خرابی در مؤلفه‌هایش تشخیص دهد
- *ping/echo* تاکتیک متداولی برای تشخیص خرابی است
- دغدغه‌های (اثرات جانبی) مرتبط با تاکتیک *ping/echo*:
  - امنیت: چگونه از وقوع حمله *ping flood* جلوگیری کرد؟
  - کارایی: چگونه می‌توان اطمینان حاصل کرد که سربار کارایی *ping/echo* کم است؟
  - قابلیت اصلاح: چگونه می‌توان *ping/echo* را به معماری موجود اضافه کرد؟

مثالی از تاکتیک‌ها و تعاملات (ادامه)

- تصمیمات معمار تاکنون به صورت زیر است

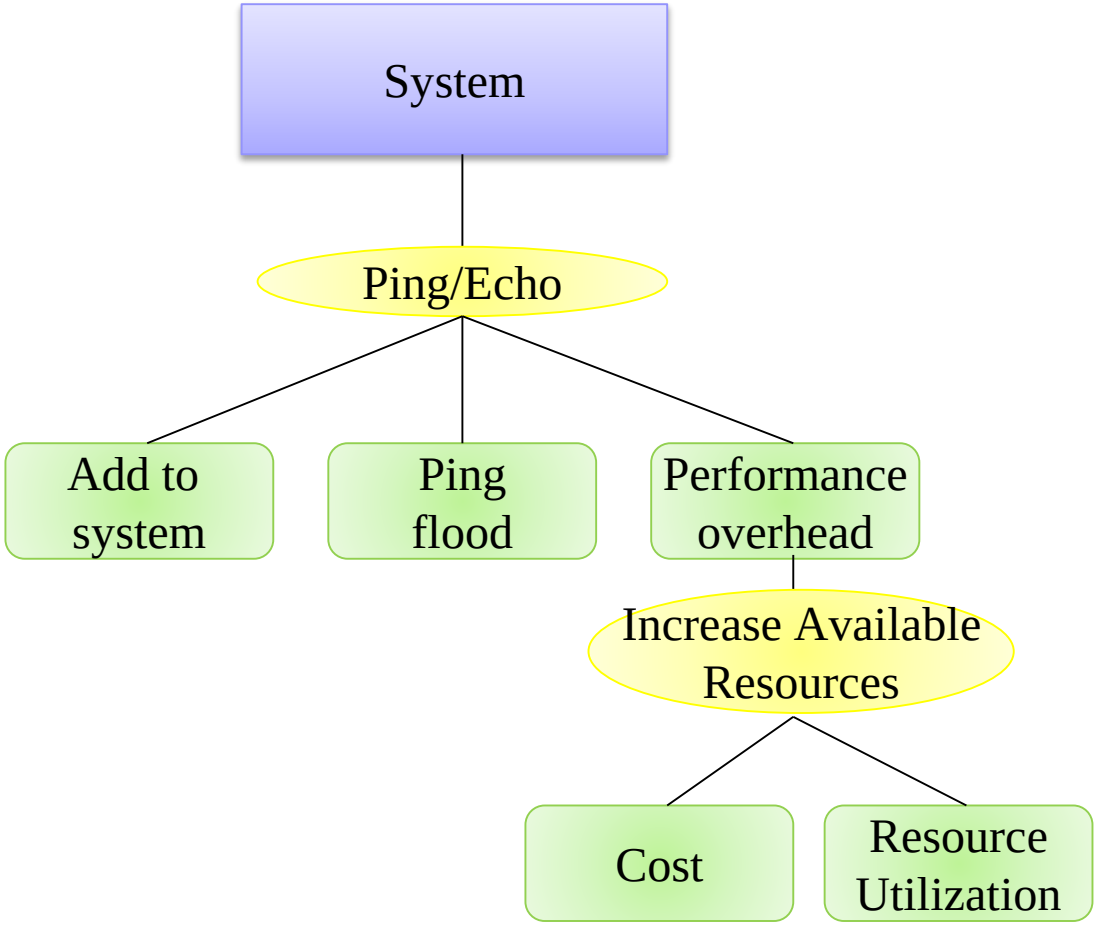


مثالی از تاکتیک‌ها و تعاملات (ادامه)

- فرض کنید معمار تشخیص می‌دهد که مصالحه کارایی (سربار ناشی از اضافه کردن *ping/echo*) مهم‌ترین است
- تاکتیک افزایش منابع در دسترس (*increase available resources*) برای مقابله با اثرات جانبی ایجاد شده بر کارایی در نظر گرفته می‌شود
- دغدغه‌های (اثرات جانبی) مرتبط با تاکتیک افزایش منابع در دسترس:
  - هزینه: افزایش منابع هزینه‌بر است
  - کارایی: نحوه بکارگیری افزایش منابع به صورت کارا و مؤثر

مثالی از تاکتیک‌ها و تعاملات (ادامه)

- مجموعه تصمیمات طراحی به شکل زیر است

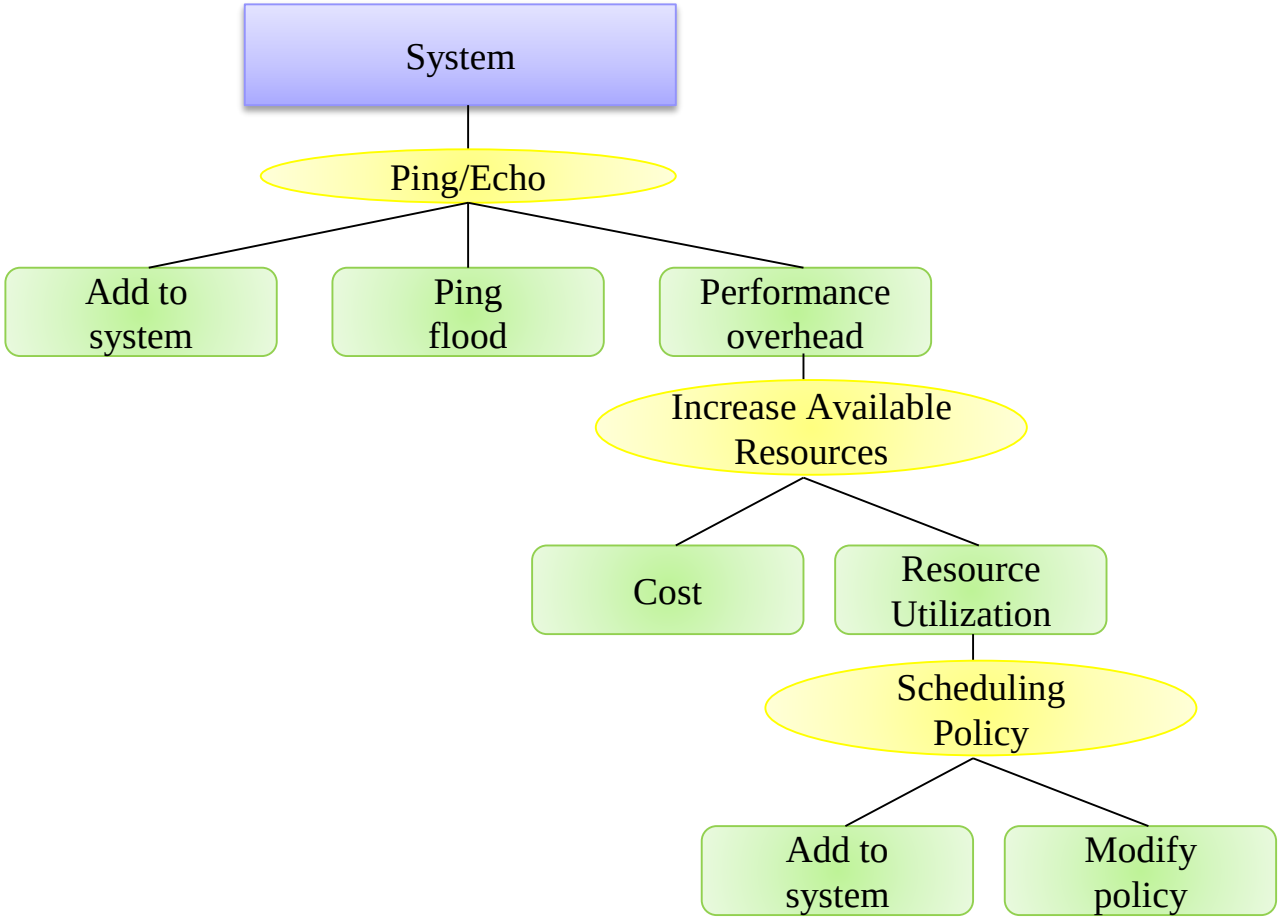


مثالی از تاکتیک‌ها و تعاملات (ادامه)

- فرض کنید معمار تصمیم می‌گیرد با اثر استفاده منابع که ناشی از بکارگیری تاکتیک افزایش منابع در دسترس (*increase available resources*) مقابله کند
- تاکتیک سیاست زمان‌بندی (*scheduling policy*) تاکتیکی است که برای استفاده کارا و مؤثر از منابع به کار برده می‌شود
- دغدغه‌های (اثرات جانبی) مرتبط با تاکتیک سیاست زمان‌بندی:
  - قابلیت اصلاح: چگونه می‌توان سیاست زمان‌بندی را به معماری موجود اضافه کرد؟
  - قابلیت اصلاح: چگونه می‌توان سیاست زمان‌بندی را در آینده تغییر داد؟

مثالی از تاکتیک‌ها و تعاملات (ادامه)

- مجموعه تصمیمات طراحی تاکنون به شکل زیر است

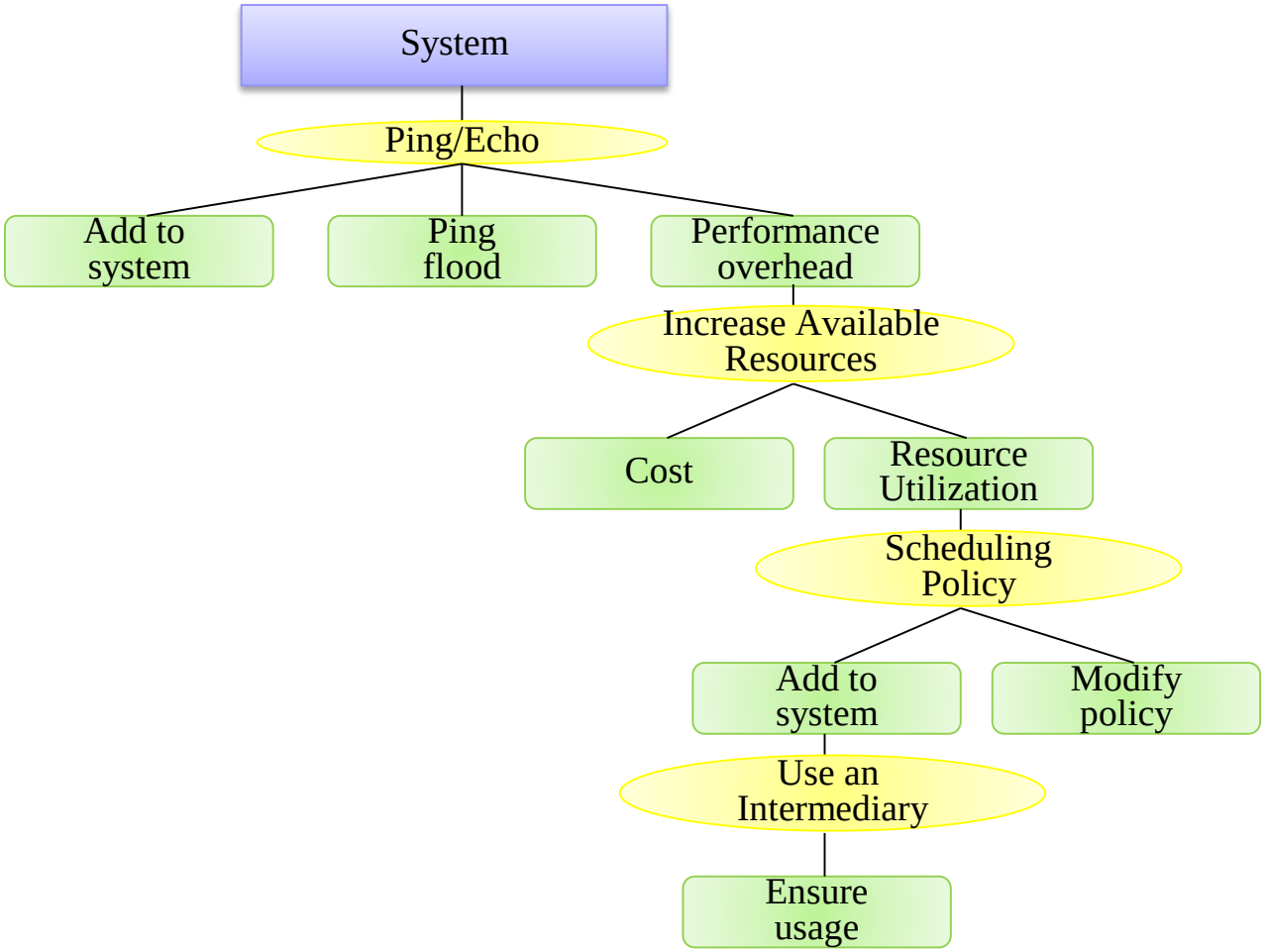


مثالی از تاکتیک‌ها و تعاملات (ادامه)

- فرض کنید معمار تصمیم می‌گیرد با اثر بر قابلیت اصلاح که ناشی از بکارگیری تاکتیک سیاست زمان‌بندی (*scheduling policy*) مقابله کند
- تاکتیک استفاده از واسط (*use an intermediary*) تاکتیکی است به افزودن زمان‌بند مرتبط می‌شود
- دغدغه‌های (اثرات جانبی) مرتبط با تاکتیک استفاده از واسط:
  - قابلیت اصلاح: چگونه می‌توان از برقراری تمام ارتباطات از طریق واسط اطمینان یافت؟

مثالی از تاکتیک‌ها و تعاملات (ادامه)

- مجموعه تصمیمات طراحی تاکنون به شکل زیر است



مثالی از تاکتیک‌ها و تعاملات (ادامه)

- تاکتیک محدود کردن وابستگی‌ها (*restrict dependencies*) به دغدغه برقراری تمامی ارتباطات از طریق واسط می‌پردازد است
- دغدغه (اثر جانبی) مرتبط با تاکتیک محدود کردن وابستگی‌ها :
  - کارایی: چگونه می‌توان اطمینان یافت که سربار ناشی از واسط بیش از اندازه نیست؟

مشکل طراحی بازگشتی شده است!

چگونه این فرایند پایان می‌یابد؟

- استفاده از هر تاکتیک دغدغه‌های جدیدی را ایجاد می‌کند
- هر دغدغه جدید موجب افزودن تاکتیک‌های جدید می‌شود

- آیا این یک تسلسل است؟

خیر، در نهایت اثرات جانبی هر تاکتیک آن قدر کم خواهد شد که قابل چشم‌پوشی خواهد بود