Тема 12. Сервіс-орієнтована архітектура

Service-Oriented Architecture (SOA)

SOA — це архітектура, у якій система складається з незалежних сервісів, що спілкуються через стандартизовані інтерфейси.

Компоненти SOA:

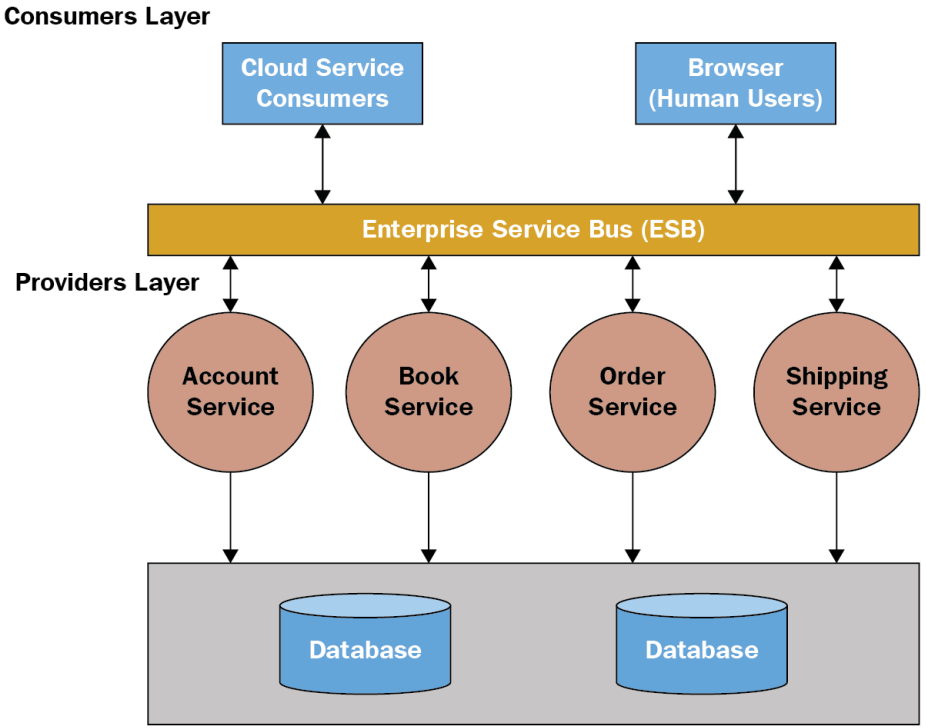
- **ESB** (Enterprise Service Bus) — маршрутизація, трансформація, доставка повідомлень.
- **Сервіси** — бізнес-функції (Order, Payment, Report).
- **Контракти** — опис запитів та відповідей.
- **Реєстр сервісів** — централізоване виявлення та управління.

Кожен сервіс має чітко визначений інтерфейс і може бути повторно використаний іншими системами.

Взаємодія відбувається через сервісну шину (ESB) або API.

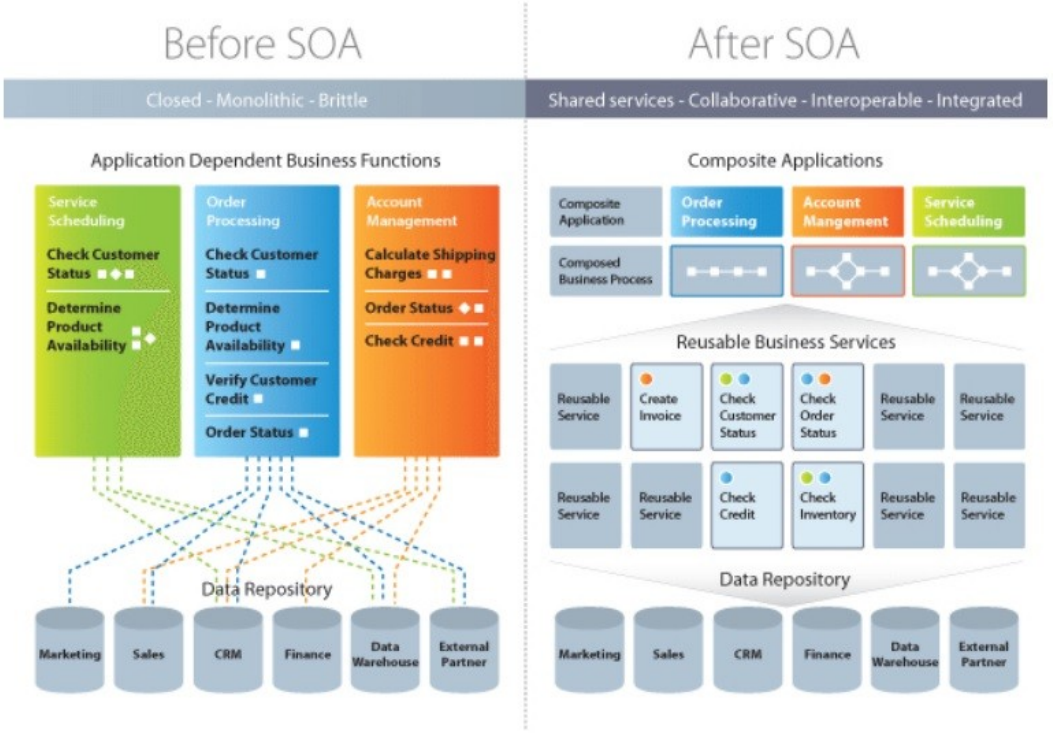
Переваги: масштабованість, повторне використання, гнучкість.

Недоліки: централізована шина може бути вузьким місцем.



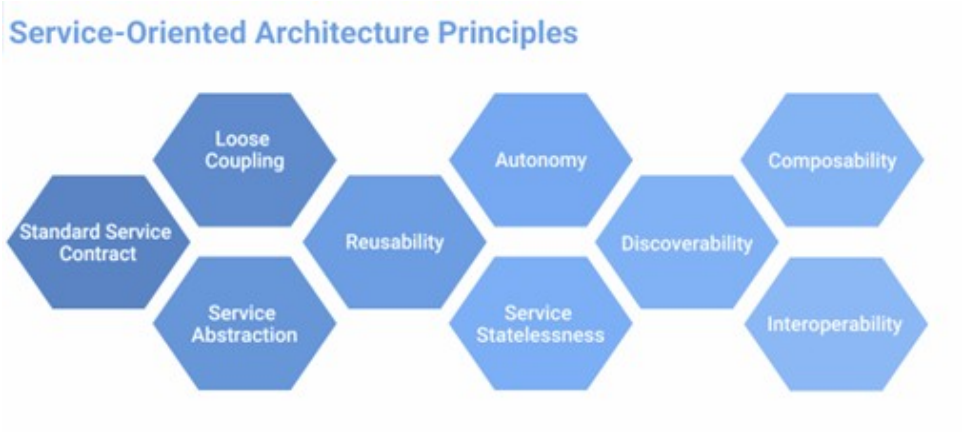
Service-Oriented Architecture (SOA)

Історично можна зобразити еволюцію архітектурних підходів наступним ланцюжком, розглянуті в темі стилі підкреслено:
Розподілені системи → REST → **SOA → Асинхронна SOA → Event-Driven Architecture** → Мікросервіси.

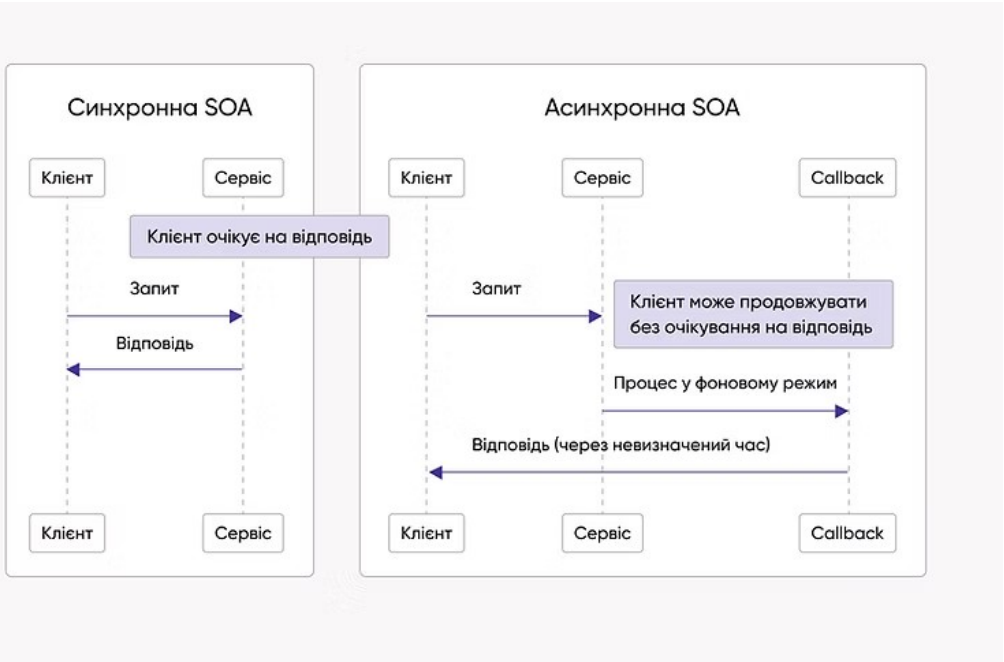


Характеристики SOA

- **Loose Coupling (слабке зв’язування)**: сервіси мінімально залежать один від одного.
- **Service Contract: чіткий інтерфейс** (наприклад, WSDL, OpenAPI).
- **Discoverability**: сервіси можуть бути знайдені через реєстр (service registry).
- **Reusability**: сервіси проектуються для повторного використання.
- **Abstraction**: внутрішня реалізація прихована за інтерфейсом.
- **Autonomy**: сервіси управляють своєю логікою і станом.
- **Composability**: сервіси можуть комбінуватися у більші бізнес-процеси.



Синхронна і асинхронна SOA



Асинхронною називають SOA, в якій кожен сервіс є автономним і виконує окреме завдання. Якщо синхронна система надсилає запит і чекає негайної відповіді, то в цьому дизайні комунікація між сервісами цього не потребує. Це означає, що клієнт може надіслати запит і перейти до інших завдань, не чекаючи відповіді.

Переваги SOA

Порівняно з архітектурами, що передували їй, SOA пропонувала значні переваги для підприємства:

- **Більша гнучкість бізнесу; швидший вихід на ринок**: ключовим є повторне використання. Ефективність складання додатків з повторно використовуваних сервісів, тобто будівельних блоків, замість переписування та повторної інтеграції з кожним новим проектом розробки, дозволяє розробникам створювати додатки набагато швидше у відповідь на нові бізнес-можливості.
- **Можливість використовувати застарілі функції на нових ринках**: Добре розроблена SOA дозволяє розробникам легко брати функції, «заблоковані» в одній обчислювальній платформі або середовищі, та розширювати їх на нові середовища та ринки. Наприклад, багато компаній використовували SOA для надання доступу до функцій фінансових систем на основі мейнфреймів новим веб-додаткам. Це дозволяє їхнім клієнтам самостійно обслуговувати процеси та інформацію, які раніше були доступні лише через пряму взаємодію зі співробітниками або бізнес-партнерами компанії.
- **Покращена співпраця між бізнесом та IT**: У SOA послуги можна визначити в бізнес-термінах (наприклад, «створити страхову пропозицію» або «розрахувати рентабельність інвестицій в капітальне обладнання»). Це дозволяє бізнес-аналітикам ефективніше співпрацювати з розробниками над важливими висновками, такими як обсяг бізнес-процесу, визначений за допомогою послуг, або бізнес-наслідки зміни процесу, що може призвести до кращого результату.
- **Ефективна підтримка**: У монолітних додатках легше створювати, оновлювати та налагоджувати невеликі сервіси, ніж великі блоки коду. Зміна будь-якого сервісу в SOA не впливає на загальну функціональність бізнес-процесу.
- **Більша адаптивність**: SOA краще адаптується до технологічних досягнень. Ви можете ефективно та економічно вигідно модернізувати свої додатки. Наприклад, організації охорони здоров'я можуть використовувати функціональність старіших систем електронних медичних записів у новіших хмарних додатках.

Недоліки SOA

- **Складність**: SOA може внести додаткову складність у проектування та реалізацію застосунку. Це пов'язано з необхідністю керування кількома сервісами та їх взаємодією.
- **Накладні витрати на продуктивність:** Використання кількох сервісів та зв'язок між ними може призвести до накладних витрат на продуктивність, особливо якщо сервіси розподілені по різних мережах або центрах обробки даних.

  Приклад: Запит на розміщення замовлення може потребувати проходження через кілька сервісів, кожен з яких додає затримку до загального часу відгуку.
- **Проблеми безпеки**: З більшою кількістю точок входу та каналів зв'язку SOA може збільшити поверхню атаки для потенційних загроз безпеці.

  Приклад: Кожен сервіс у SOA може мати свій власний набір вразливостей, що ускладнює захист усієї системи.
- **Управління залежностями**: Управління залежностями між сервісами може бути складним, особливо зі зростанням кількості сервісів.

  Приклад: Якщо один сервіс зазнає простою, це може вплинути на інші сервіси, які від нього залежать, що призведе до каскадного збою.
- **Вартість**: Впровадження та підтримка SOA може бути дорогим через потребу в спеціалізованих інструментах, навчанні та інфраструктурі.

  Приклад: Організаціям може знадобитися інвестувати в сервісні шини, шлюзи API та інструменти моніторингу для ефективного управління своєю SOA.

Приклади SOA

До 2010 року впровадження SOA на повну потужність йшло у провідних компаніях практично в кожній галузі. Наприклад:

- Компанія Delaware Electric звернулася до SOA для інтеграції систем, які раніше не взаємодіяли одна з одною, що призвело до підвищення ефективності розробки, яка допомогла організації залишатися платоспроможною протягом п'ятирічного, державного заморожування тарифів на електроенергію.
- Cisco впровадила SOA, щоб забезпечити узгодженість процесу замовлення продуктів для всіх продуктів і каналів, представивши процеси замовлення як послуги, які підрозділи, придбання та бізнес-партнери Cisco можуть інтегрувати на свої веб-сайти.
- Independence Blue Cross (IBC) у Філадельфії впровадила SOA, щоб забезпечити роботу різних учасників, які мають справу з даними пацієнтів — агентів служби підтримки клієнтів IBC, лікарень, користувачів веб-сайту IBC — з одним і тим самим джерелом даних («єдиним джерелом достовірної інформації»).

Джерело https://www.ibm.com/think/topics/soa

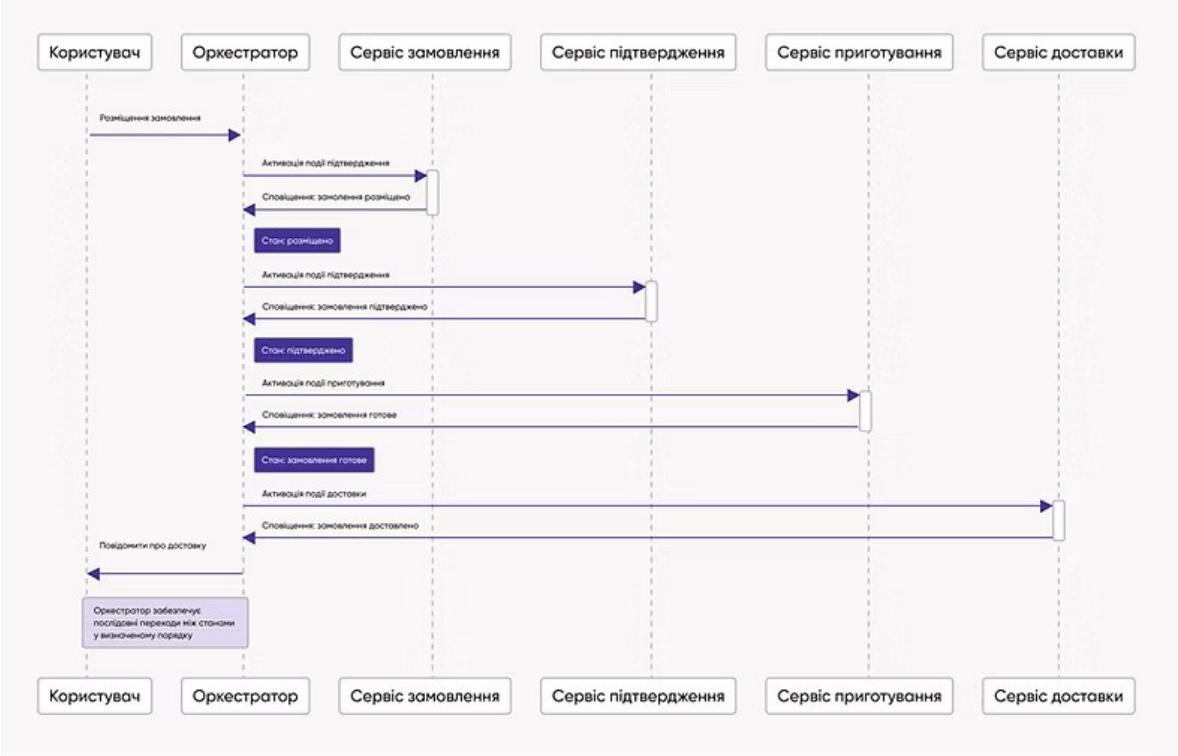
Приклади SOA

До 2010 року впровадження SOA на повну потужність йшло у провідних компаніях практично в кожній галузі. Наприклад:

- Компанія Delaware Electric звернулася до SOA для інтеграції систем, які раніше не взаємодіяли одна з одною, що призвело до підвищення ефективності розробки, яка допомогла організації залишатися платоспроможною протягом п'ятирічного, державного заморожування тарифів на електроенергію.
- Cisco впровадила SOA, щоб забезпечити узгодженість процесу замовлення продуктів для всіх продуктів і каналів, представивши процеси замовлення як послуги, які підрозділи, придбання та бізнес-партнери Cisco можуть інтегрувати на свої веб-сайти.
- Independence Blue Cross (IBC) у Філадельфії впровадила SOA, щоб забезпечити роботу різних учасників, які мають справу з даними пацієнтів — агентів служби підтримки клієнтів IBC, лікарень, користувачів веб-сайту IBC — з одним і тим самим джерелом даних («єдиним джерелом достовірної інформації»).

Джерело https://www.ibm.com/think/topics/soa

Приклади SOA



Приклад асинхронної SOA архітектури сервісу доставки їжі з оркестратором

Архітектура, керована подіями
Event-Driven Architecture (EDA)

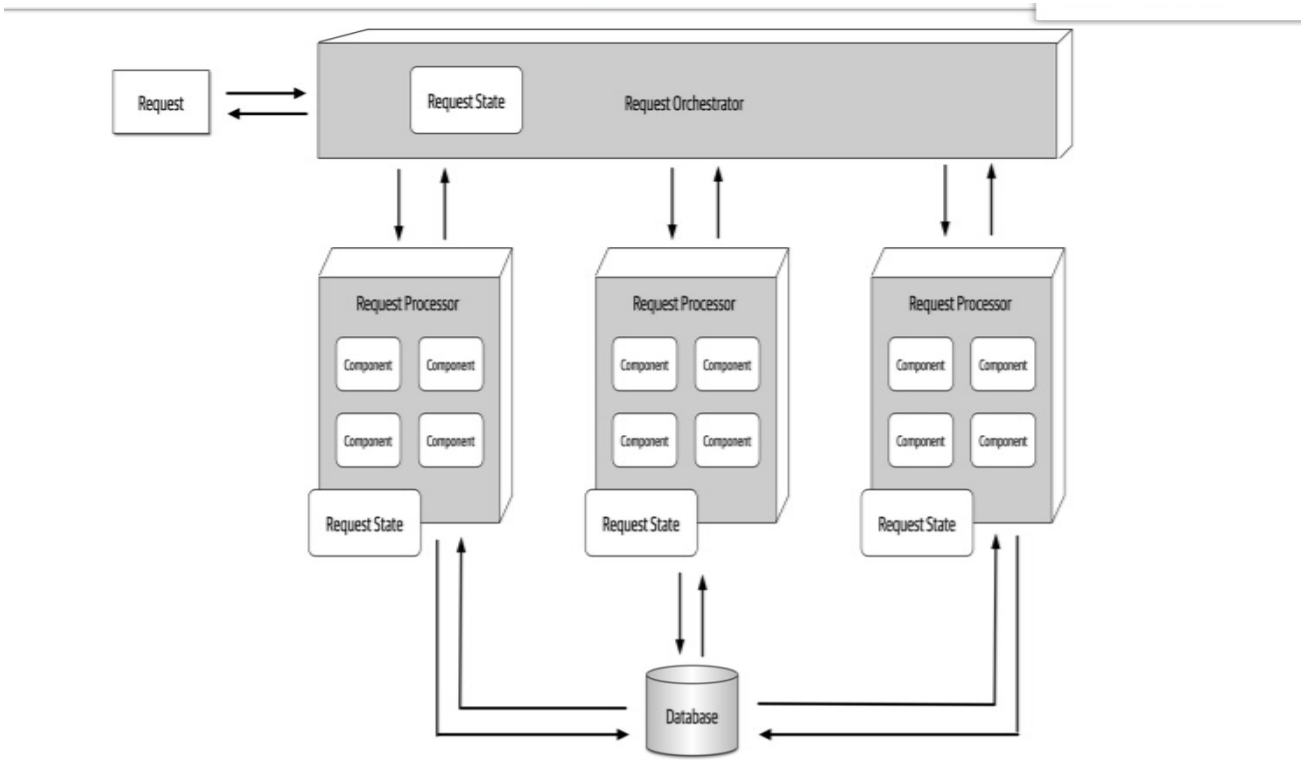
Архітектура, керована под

- EDA — це парадигма, де події - головна одиниця взаємодії.
- Подія — щось, що сталося (наприклад: OrderCreated, PaymentReceived).
- Сервіси публікують події без знання про споживачів.
- Інші сервіси підписуються на події, реагують потом.
- EDA — це природне продовження асинхронної SOA, де подія є головною одиницею взаємодії.

Архітектура, керована под

- EDA — це парадигма, де події - головна одиниця взаємодії.
- Подія — щось, що сталося (наприклад: OrderCreated, PaymentReceived).
- Сервіси публікують події без знання про споживачів.
- Інші сервіси підписуються на події, реагують потом.
- EDA — це природне продовження асинхронної SOA, де подія є головною одиницею взаємодії.

Архітектура, керована подіями



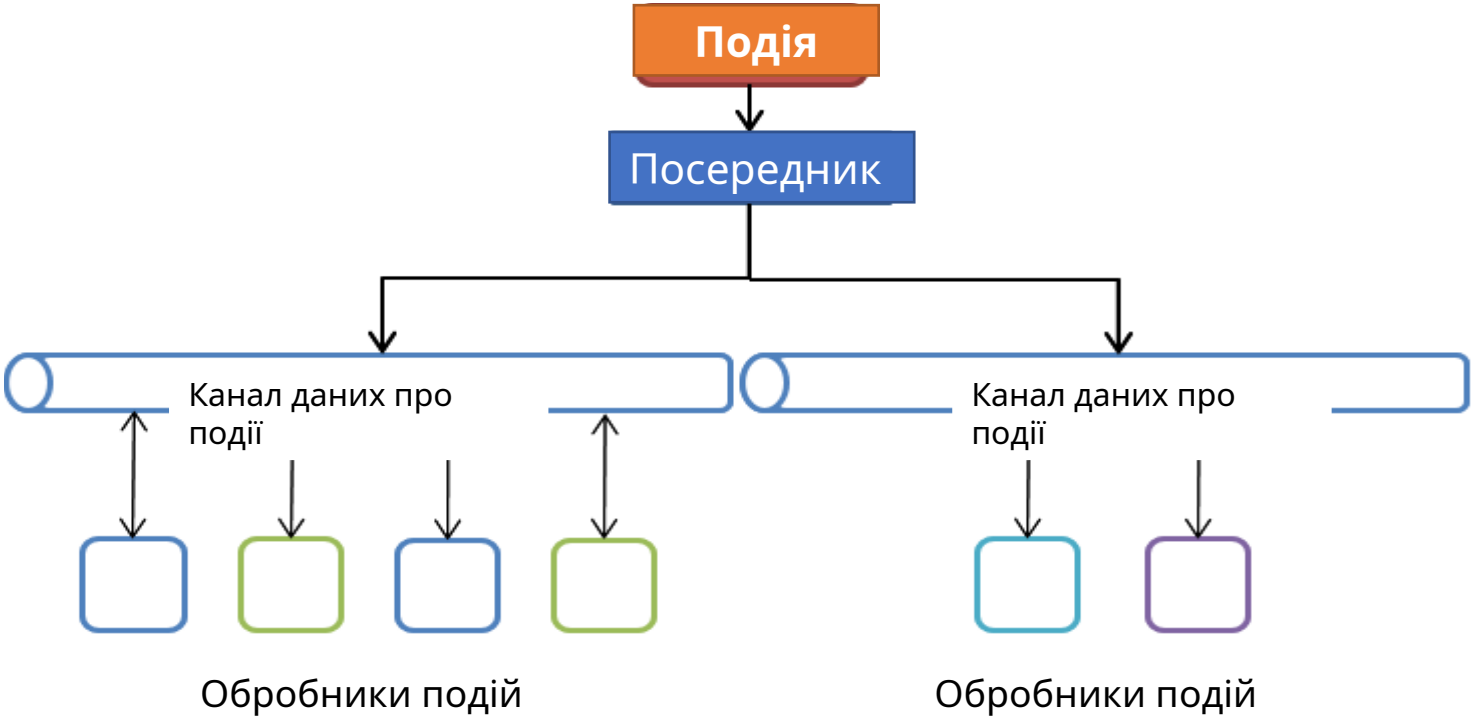
Більшість додатків дотримуються того, що називається моделлю на основі запиту. Хорошим прикладом моделі на основі запиту є запит від клієнта, щоб отримати історію замовлень за останні півроку.
Отримання інформації історії замовлень є керованим даними, детермінованим запитом, який зроблений до системи даних у конкретному контексті, а не подія, на яку повинна реагувати система.

Архітектура, керована под

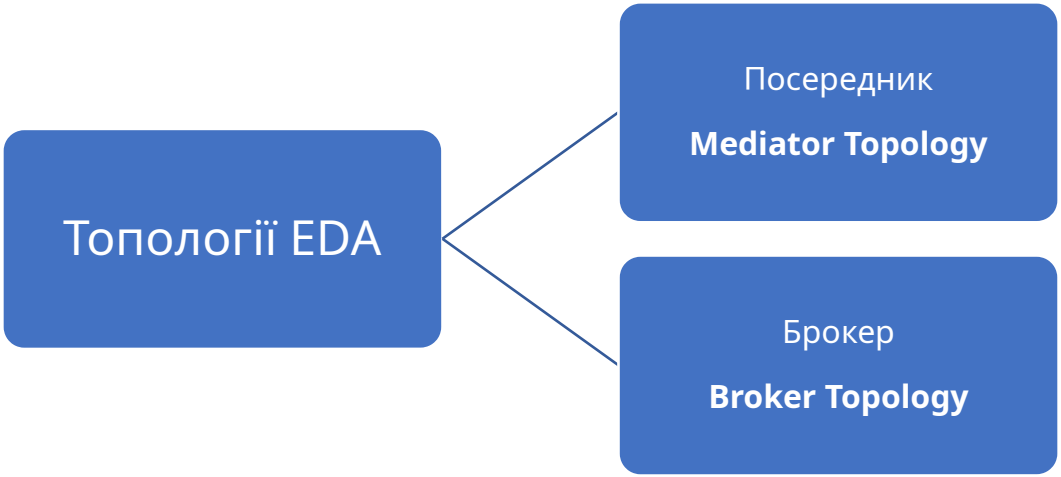
Якщо говорити про програмне забезпечення, то в цій схемі існує два варіанти подій: ініціююча подія і подія, на яку реагують обробники. Обробники є ізольованими незалежними компонентами, що відповідають (в ідеалі) за якусь одну задачу, і містять бізнес-логіку, необхідну для роботи.

- **Посередник може бути** реалізований декількома способами. Самий просто спосіб - це скористатися фреймворками для інтеграції Apache Camel (https://camel.apache.org/), Spring Integration (https://docs.spring.io/spring-integration/reference/html/index.html) або Mule ESB (https://www.mulesoft.com/resources/esb/what-mule-esb). Для великих додатків, яким потрібна більш складні функції управління, ви можете реалізувати посередника, використовуючи концепцію управління бізнес-процесами (наприклад рушій jBPL) https://www.redhat.com/en/technologies/jboss-middleware/process-automation-manager.
- **Архітектура, керована подіями** - це відносно складний патерн. Причиною тому - його розподілена і асинхронна природа. Вам доведеться вирішувати проблеми фрагментації мережі, обробляти помилки в черзі подій і так далі. Плюсами цієї архітектури можуть служити висока продуктивність, легкість розгортки і вражаючі можливості масштабування. Однак можливо ускладнення процесу тестування системи.

Архітектура, керована подіями



Архітектура, керована подіями



EDA складається з **двох основних топологій, посередник і брокер**.

- Топологія посередник зазвичай використовується, коли вам потрібно організувати кілька кроків у події через центрального посередника, в той час як
- топологія брокера використовується, коли ви хочете організувати ланцюг подій разом без використання центрального посередника.

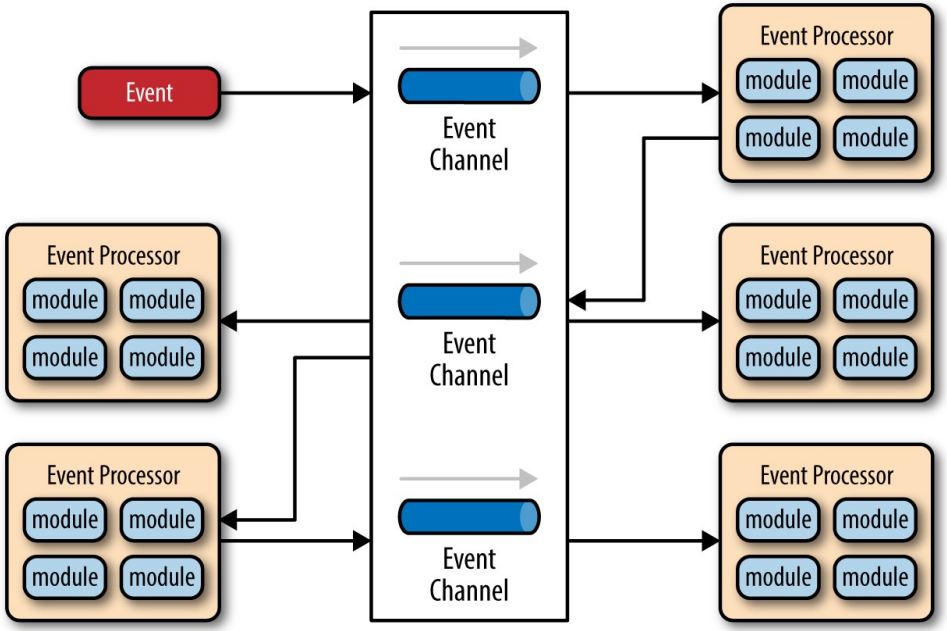
Оскільки характеристики і стратегії здійснення розрізняються між цими двома топологіями, важливо розуміти, кожну з них, щоб знати, яка найкраще підходить для вашої конкретної ситуації.

Топологія Брокер

Топологія брокер відрізняється від топології посередника в тому, що немає ніякого центрального посередника подій; а потік повідомлень розподіляється між компонентами процесора подій в каналах через прості повідомлення брокера. Ця топологія корисна, якщо у вас є відносно простий потік обробки подій, і ви не бажаєте (або вам не потрібно) узгодженість подій через єдиний центр.

Є два основних типи компонентів архітектури в рамках топології брокера: компонент брокера і компонент процесора подій. Компонент брокер може бути централізованим або федеративним і містить всі канали подій, які використовуються в потоці подій. Канали подій, що містяться в компоненті брокера можуть бути чергами повідомлень, темами повідомлень, або комбінацією обох.

Топологія Брокер



Як ви можете бачити з діаграми, немає ніякого центрального компонента подій посередника управління і узгодження початкового події; швидше, кожен компонент процесор подій відповідає за обробку події та публікації нової події, вказуючи дію, яку він щойно виконав. Наприклад, процесор подій, який врівноважує портфель акцій може отримати початкову подію під назвою «дроблення акцій». На підставі цієї початкової події, процесор подій може зробити деяку зміну балансу портфеля, а потім опублікувати нову подію для брокера під назвою “ перебалансування портфеля ”, яка потім буде підібрана іншим процесором подій. Зверніть увагу, що бувають випадки, коли подія публікується процесором подій, але не підбирається будь-яким процесором іншої події. Це часто зустрічається, коли додаток еволюціонує або для забезпечення майбутньої функціональності і розширень.

Плюси/мінуси брокера

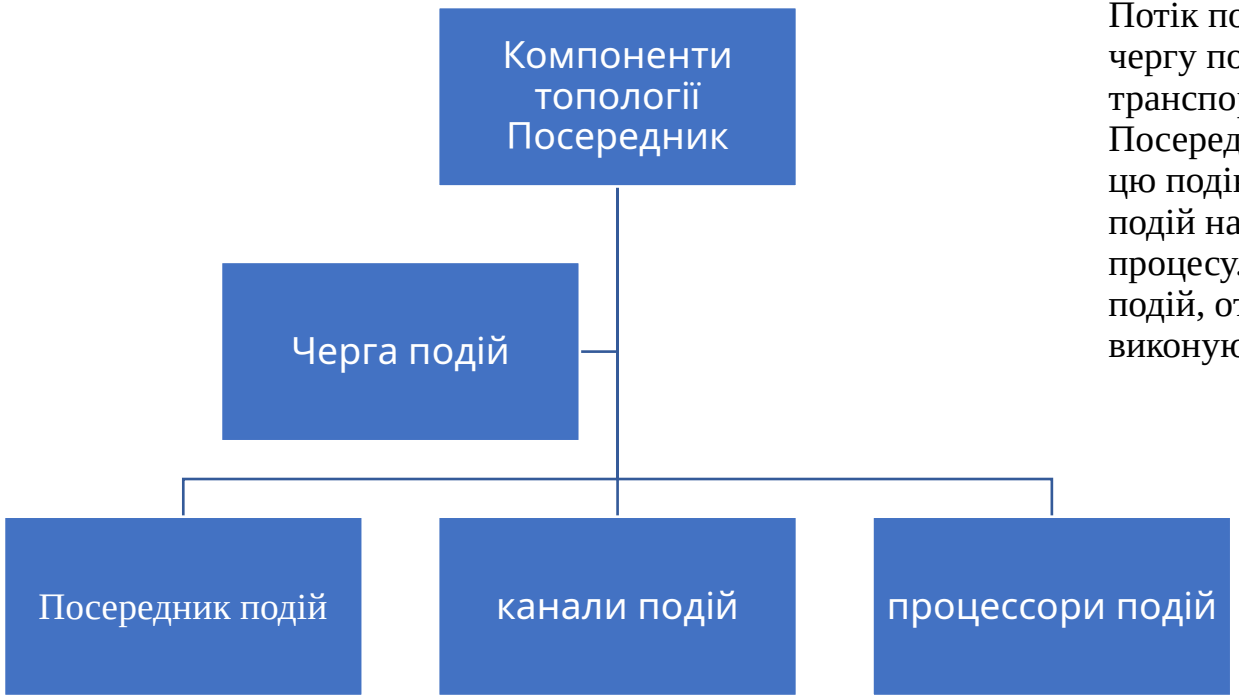
Переваги	Недоліки
Сильно роз'єднані процесори подій	управління робочим процесом
Висока масштабованість	Помилки обробки
Висока гнучкість(High responsiveness)	Складне відновлення (Recoverability)
Висока працездатність	Перезавантаження можливостей (Restart capabilities)
Висока відмовостійкість	Невідповідність даних

Топологія Посередник

Топологія посередник використовується для подій, які мають кілька кроків і вимагають деякого рівня узгодженості для обробки події. Наприклад, одна подія для розміщення біржової торгівлі може зажадати, щоб спочатку перевірили угоду, а потім перевірили відповідність цієї біржової торгівлі щодо дотримання різних правил, призначити торгівлю брокеру, порахувати комісію, і, нарешті, місце торгівлі з брокером.

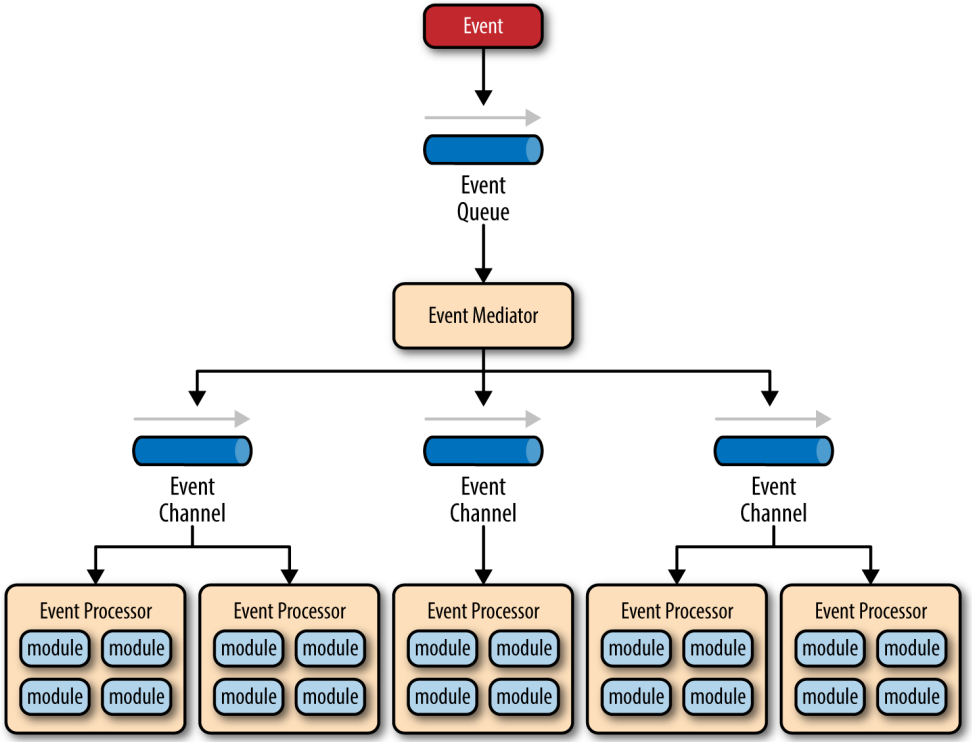
Всі ці кроки зажадають деякого рівня узгодженості, щоб визначити порядок кроків і які з них можна робити послідовно а які паралельно.

Топологія Посередник



Потік подій починається з клієнта посилає подію в чергу подій, яка використовується для транспортування події до посередника подій. Посередник події отримує вихідну подію і погодить цю подію шляхом відправки додаткових асинхронних подій на канали подій, щоб виконати кожен крок процесу. Процесори подій, які очікують на каналах подій, отримавши подія від посередника подій і виконують певну бізнес логіку для обробки події.

Топологія Посередник

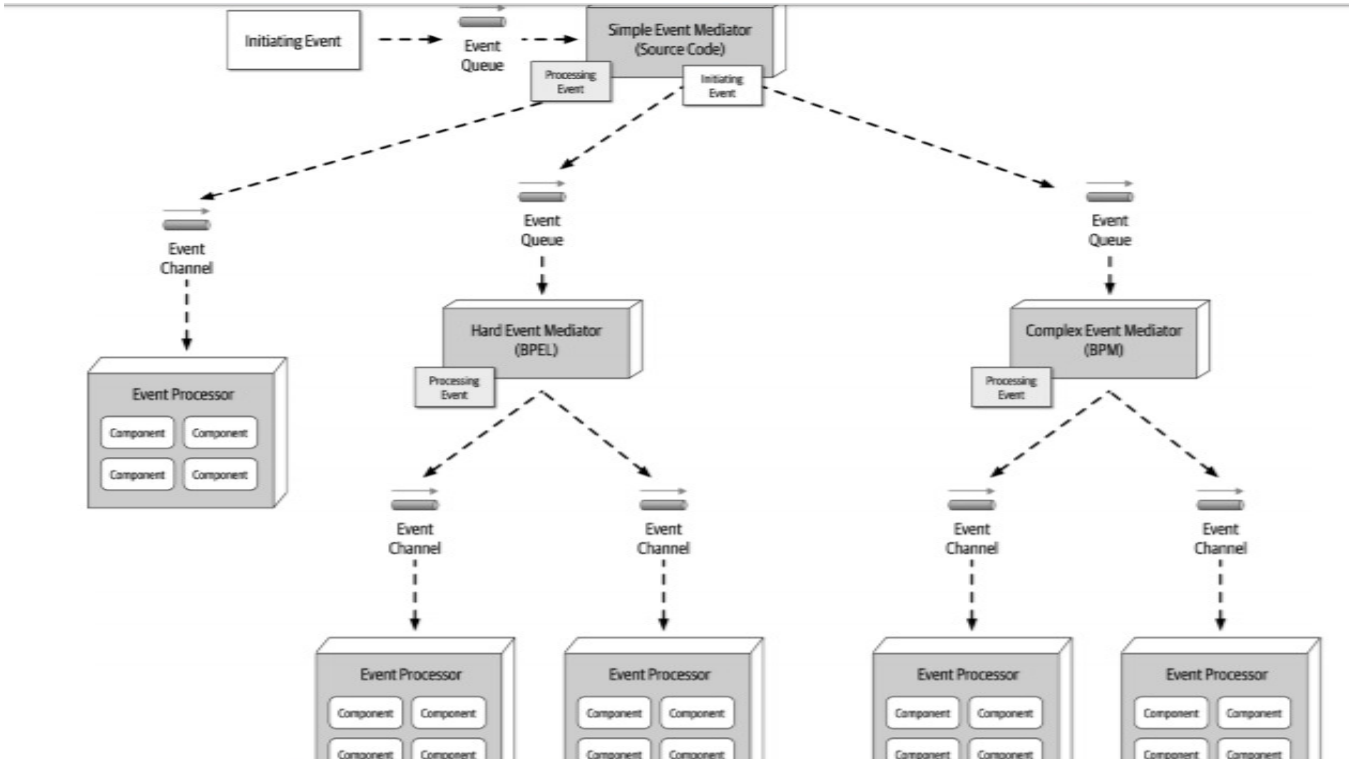


EDA

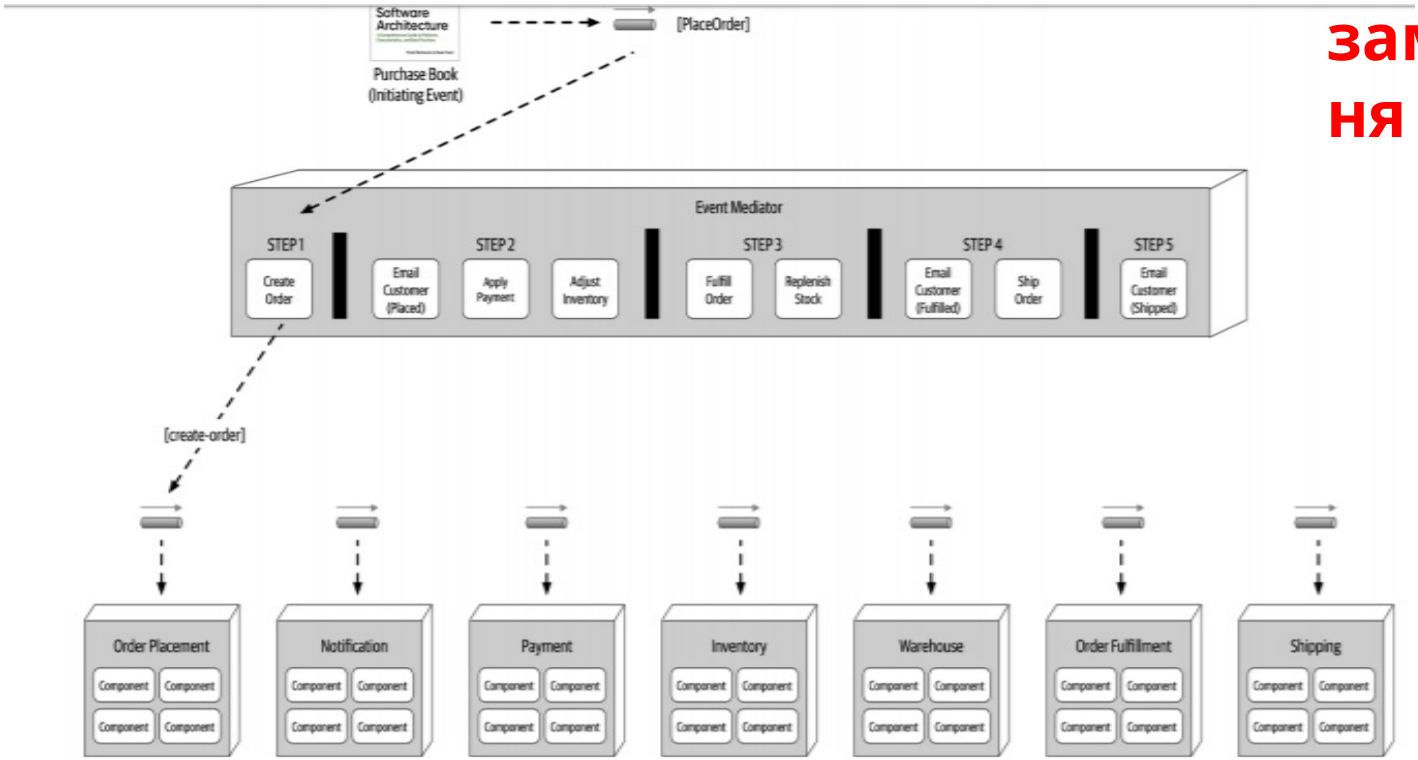
Звичайно є від десятків до декількох сотень черг подій. Шаблон не визначає реалізацію компонента «черга подій»; це може бути черга повідомлень, веб-сервіс, або будь-яка їх комбінації.

Посередник-подія може бути реалізований різними способами. Як архітектор, ви повинні розуміти, кожен з цих варіантів здійснення, щоб гарантувати, що рішення, яке ви вибираєте для посередника подій відповідає вашим потребам і вимогам.

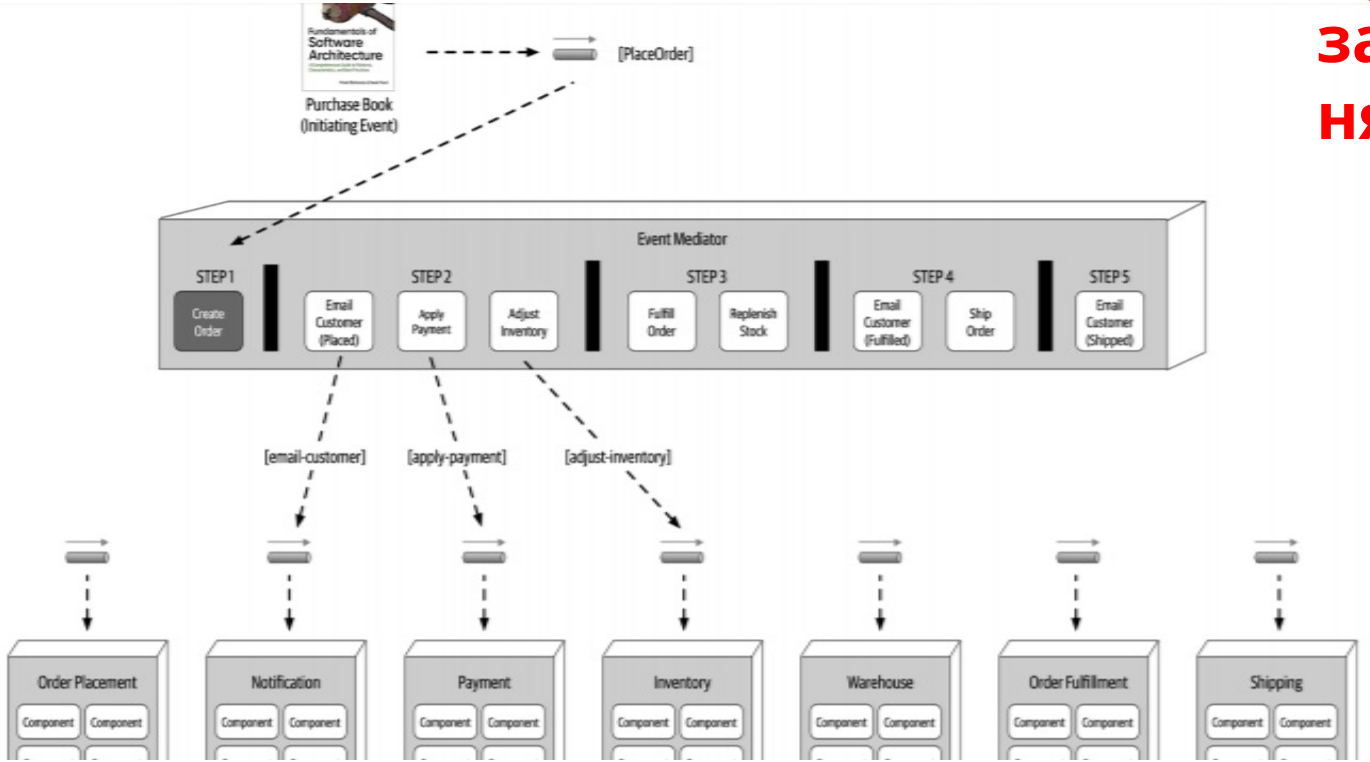
Делегування події на відповідний тип посередника подій



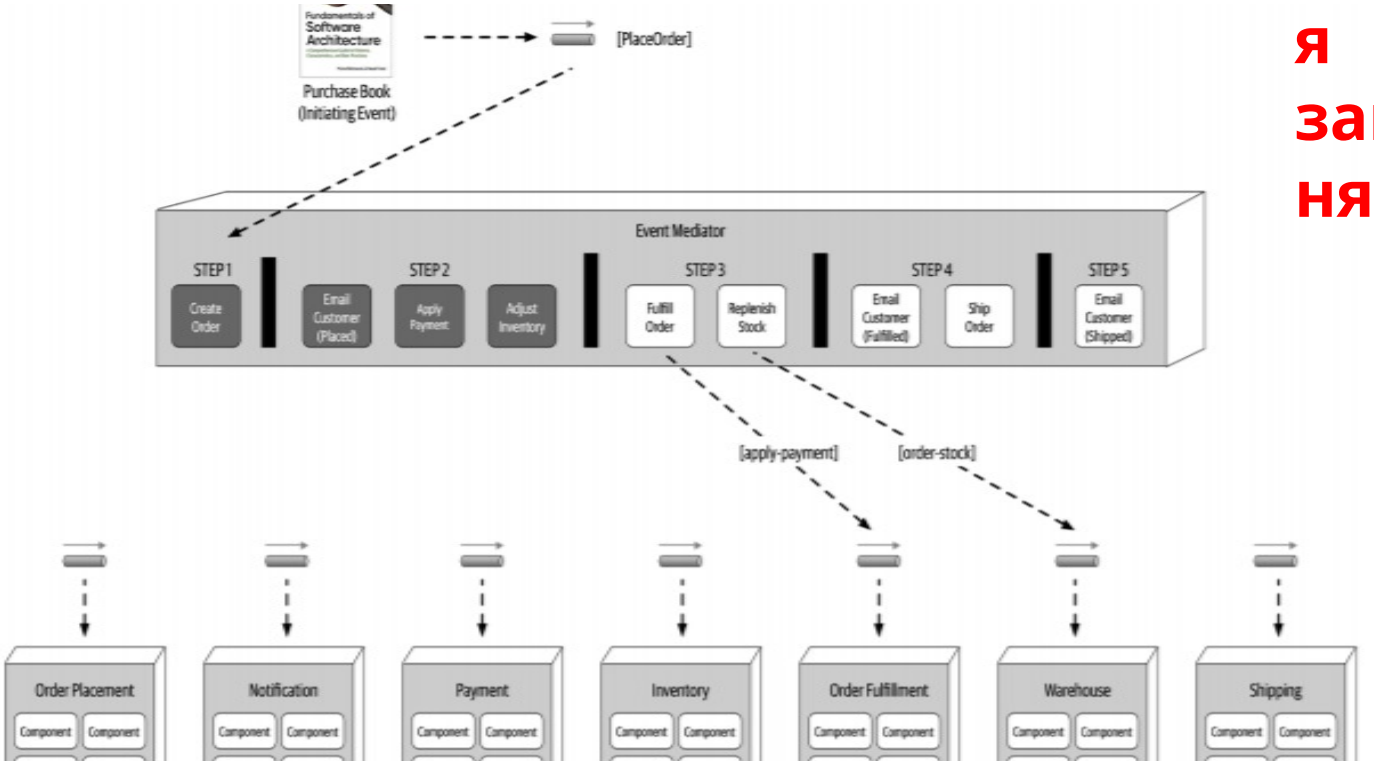
Крок 1. Зробити замовлення



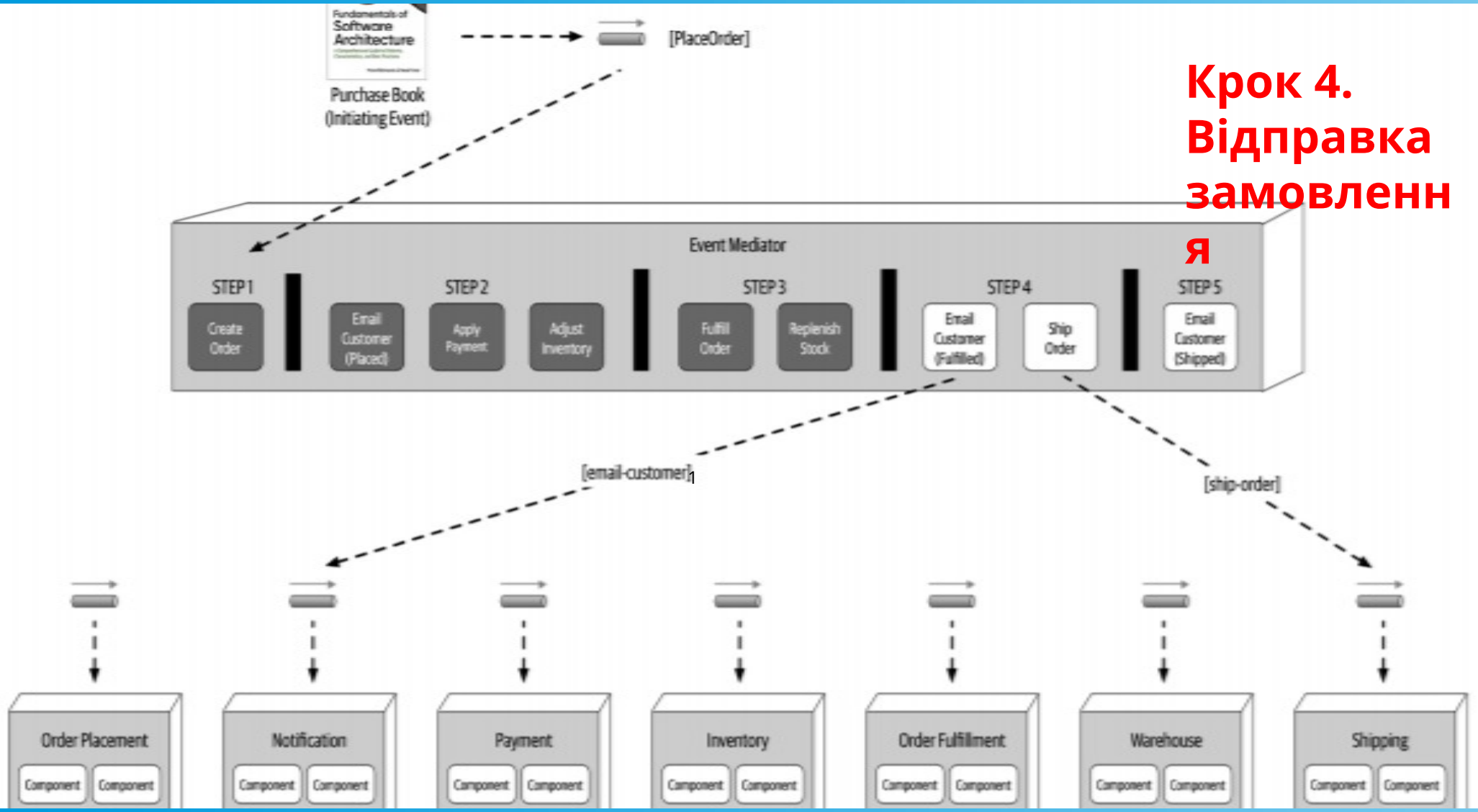
Крок 2. Процес замовлення



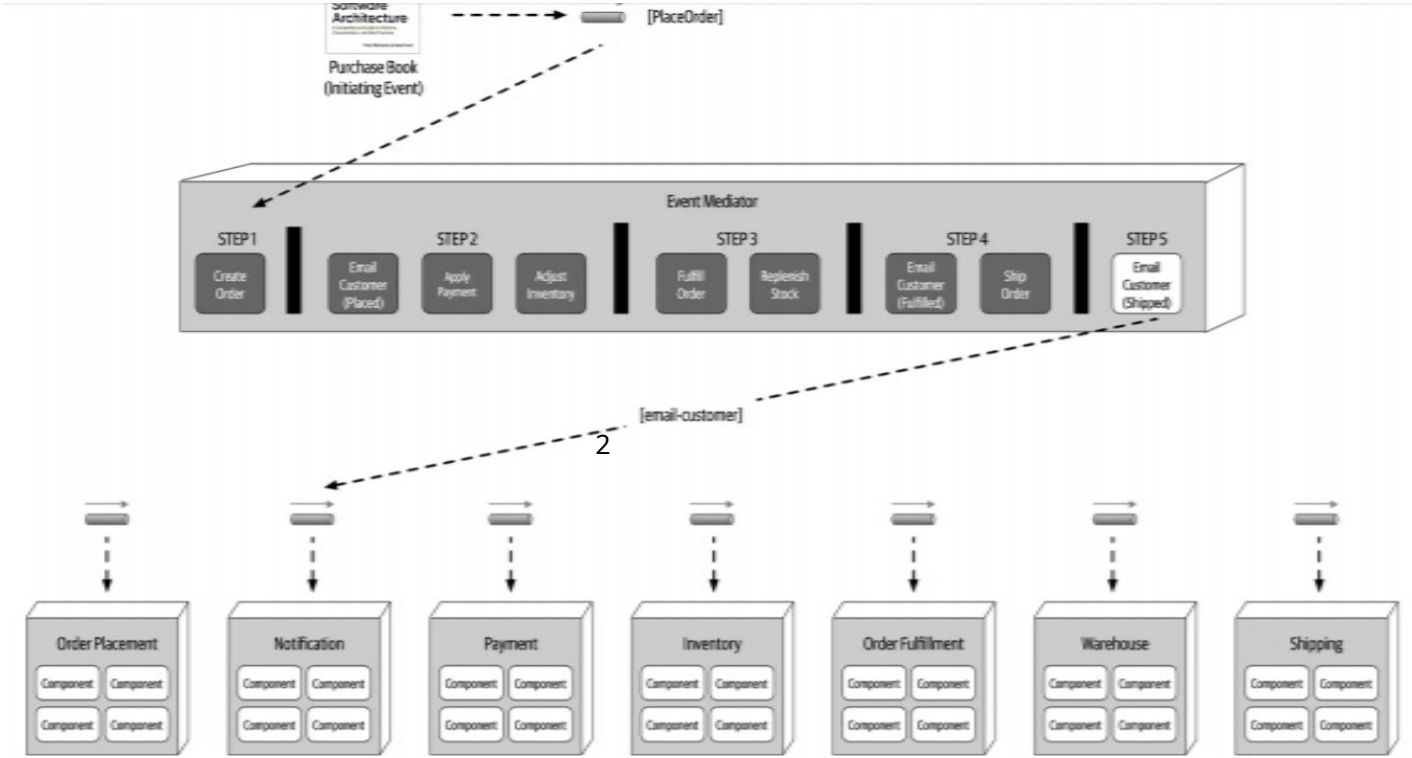
Крок 3. Виконання замовлення



Крок 4. Відправка замовлення



Крок 5. Повідомлення замовника



Плюси/мінуси посередника

Переваги	Недоліки
Контроль робочого процесу	Сильна зчеплиність процесорів подій
Обробка помилок	Менша масштабованість
Відновлення	Зниження продуктивності
Перезавантаження можливостей (Restart capabilities)	Зниження відмовостійкості
Краща узгодженість даних	Моделювання складних робочих процесів

Плюси/мінуси

Керована подіями моделі архітектура є відносно складною для реалізації моделлю, в першу чергу через її асинхронний розподілений характеру. При реалізації цього шаблону, ви повинні вирішувати різні питання, розподіленої архітектури, такі як віддалений доступ процесу, відсутність чуйності і брокер зміни логіки відмови в разі брокера або посередника.

Однією зі складностей даного патерну, є відсутність атомарних операцій для одного бізнес-процесу. Оскільки компоненти процесора подій сильно розв'язані і розподілені, то дуже важко підтримувати транзакційну єдність роботи з ними. З цієї причини при розробці вашого застосування, використовуючи цей шаблон, ви повинні постійно думати про те, які події можуть і не можуть працювати незалежно один від одного і планувати деталізацію ваших процесорів подій відповідно.

Можливо, одним з найбільш складних аспектів подієвого шаблону архітектури є створення, підтримка і управління контрактами компонентів подій процесора. Кожна подія, як правило, має конкретну інформацію, пов'язану з нею(наприклад, значення даних і формат даних, що передаються по відношенню до процесора подій). Це життєво важливо при використанні цього шаблону вибрати стандартний формат даних (наприклад, XML, JSON, об'єкт Java і т.д.) і розробити політику договору управління версіями з самого початку.

Дякую за увагу!