Software Architecture

내 시스템과 다른 시스템과 커뮤니케이션 시 필요한 설계!
설계가 필요한 커뮤니케이션이 있음.

Implicit Asynchronous Communication Software Architecture

Eunmi Choi
Kookmin University

Implicit invocation architectures

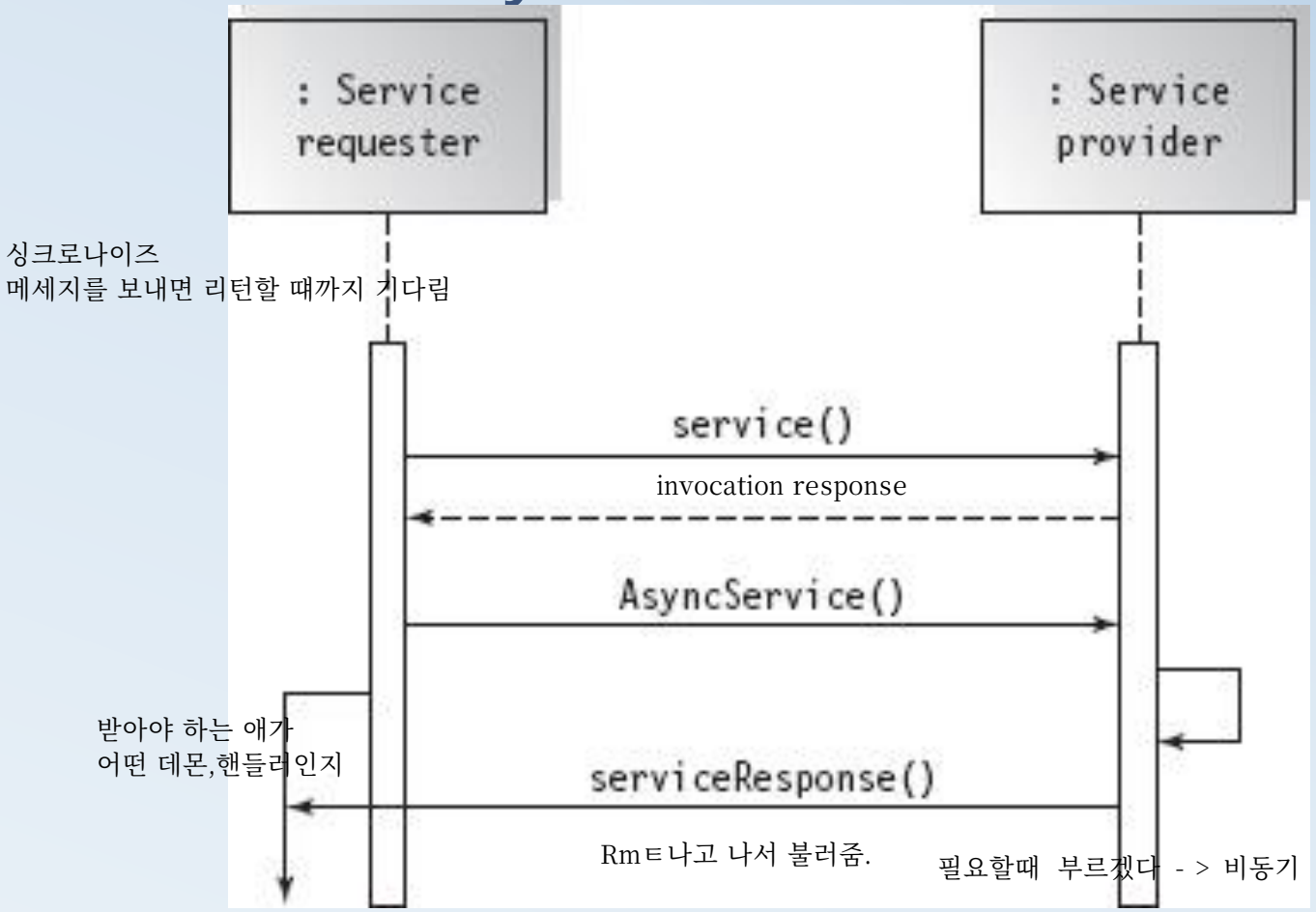
역할이 데이터를 보내는 쪽과 받는 쪽
연결이 어떻게 몇개씨 되는가에 대한 결정

센더, 리시버로 파티를 나눔.
퍼블리셔 : 의미있는 컨텐트ㅡㄹ 만드는 생산자
프로듀서 : 만드는 쪽
암튼 이 둘 간에 어케 통신할지!

이벤트 - ? / 메세지 - ?
설계할 때 어떤 의미가 있는지?

- Structure
  - Break the system into two parties:
    - one party a publisher, sender, producer, source, or any other titles for event or message announcement,
    - the other a subscriber, receiver, consumer, target, or any other title with the same meaning for registering their interest and handling the events or messages.
  - The communication between these two parties can be synchronous or asynchronous in one of the following mode
    - one-to-one (message queue), 이거 읽으심
    - one-to-many (message topic, event-based),
    - or many-to-one (event-based)

Synchronous vs. Asynchronous Invocations



싱크로나이즈
메세지를 보내면 리턴할 때까지 기다림

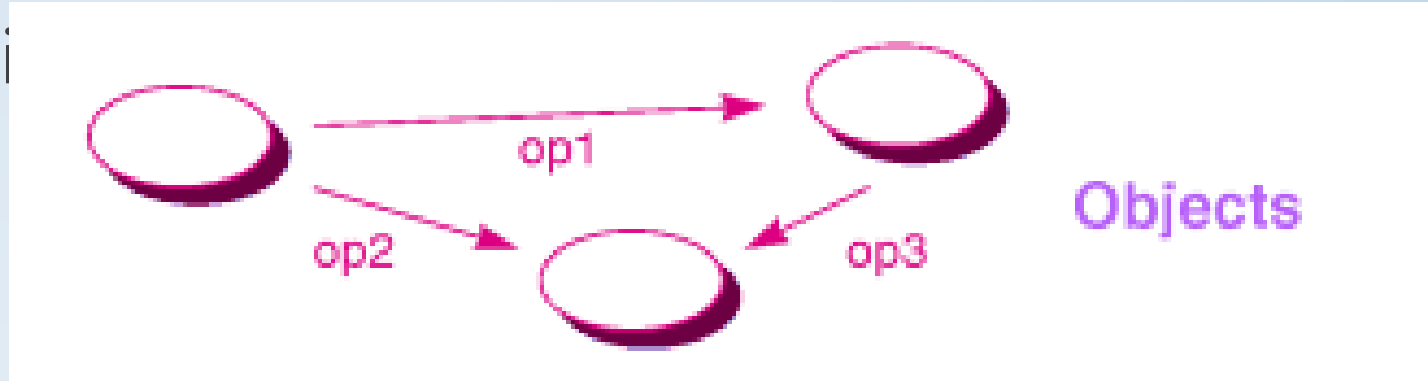
받아야 하는 애가
어떤 데몬,핸들러인지

Rmㅌ나고 나서 불러줌.

필요할때 부르겠다 - > 비동기

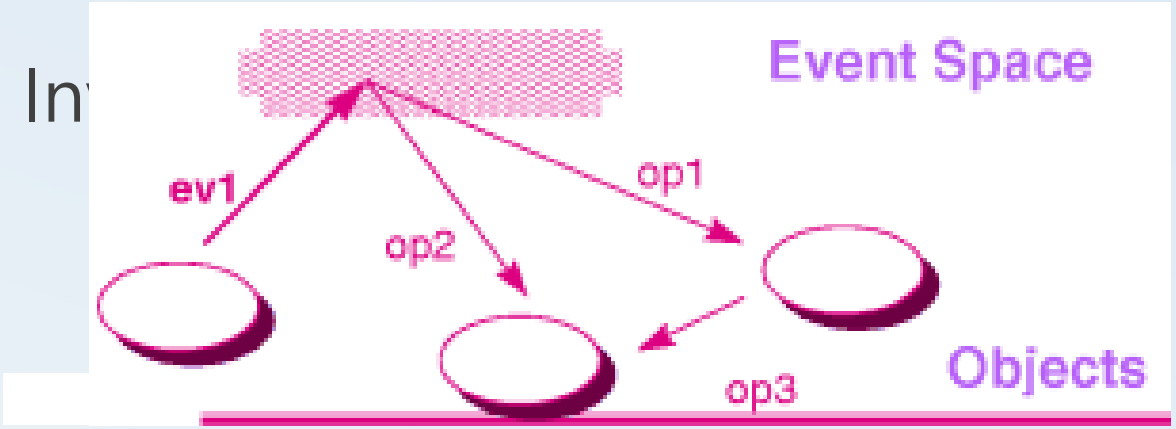
Explicit vs. Implicit Invocation

- Explici



부르고 부르고 하는걸 다 표현

- Implicit In



내재되어 잇음.
이벤트가 일어나면 그게 의미하는게 무엇이다 라는 것.

Implicit Asynchronous Communication Architecture

- Publish and

이벤트마다 그것을 처리할 수 있는 핸들러를 둠

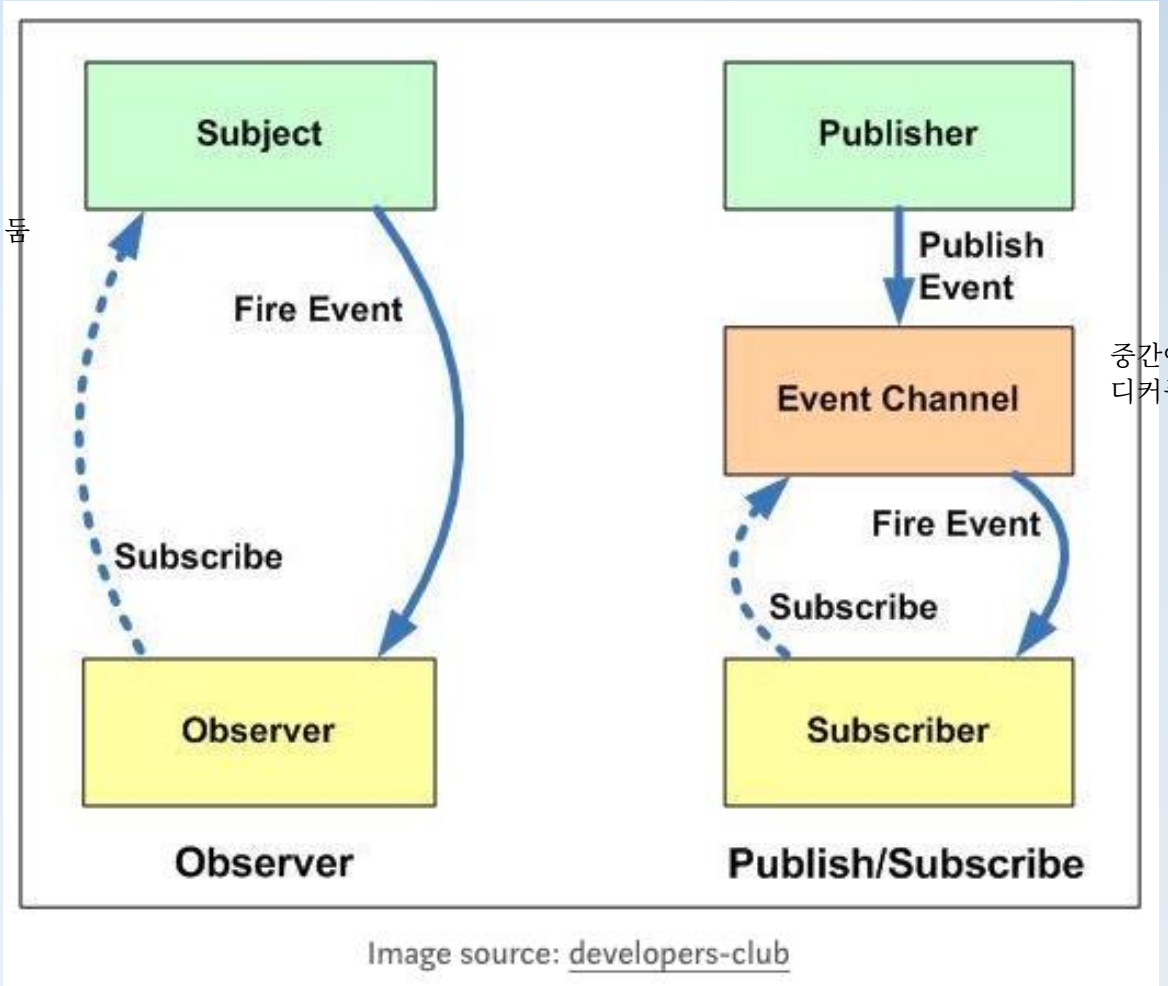


Image source: developers-club

중간에 버퍼를 두는 채널을 사용해 디커플드 시킴.

Types of Implicit Asynchronous Communication

- An asynchronous implicit invocation communication can be specified in two different modes:
  - Non-buffered Event-based
    - ex) Observer, Event Listener
  - Buffered Messaged-based
    - ex) Message Queue(One-to-One)
    - Message Topic(One-to-Many)

이건 버퍼를 쓰는 방법과 아닌 방법이 있음.

트리거가 금방 됨.
개굴 - > 개굴 처리 가능 핸들러, 리스너가 있음.

들어오는 리퀘스트와 핸ㄷㄹ러를 해주는 구조
데이터 스트럭쳐가 잇어 담고 이씅ㅁ - 많이 쓰는 구조가 큐 FIFO
큐가 가장 일반적인 메세지를 핸들해줄 수 있는 버퍼
또 다른 것은 토픽 별로 둘 수 있음. 버퍼를 하나만 두지 않고 뉴스 날씨 미세먼지 이렇게 나눠 넣을 수 있음. 뉴스에 관련된거만 볼 수 있고, 날씨에 대한 것만 볼 수 있는 것. 시간별로 상황별로 리트리뷰트 할 수 있음.

버퍼의 캐퍼시티가가장 큰 문제!! 상황에 맞춰 튜닝해야 함.

Implicit Asynchronous Communication Software Architecture

구조 자체는 누가 등록하고 구독하는 publish - subscriber 구조 또는 procducer consumer 패턴을 가짐

- Architectures that apply the publisher-subscriber or producer-consumer patterns
  - where the subscribers/consumers are interested in some events or messages issued by a publisher/producer.
  - Subscribers register themselves with the event source.
  - The subscriber is actually an event listener that, after registration, is notified of such occurrences.
  - Once an event is fired off by an event source, all corresponding subscribers are notified, which then take corresponding actions.
  - The subscribers decide on the actions to execute.
  - The Observer pattern is another name.

publising - subscribing : 출판 - 구독
producing 생산 - consuming 소비
같은 형태

Implicit Asynchronous Communication Software Architecture

- The message queue and message topic are typical buffered asynchronous architectures
  - subscribers/consumers also need to register their interests with
  - the event/message is fired off when available on the buffered message queue or topic.
  - A message queue is a one-to-one or point-to-point architecture between message senders and message receivers.
  - a message topic is a one-to-many architecture between publishers and subscribers.

퍼블리셔 : 출판
뉴스가 나왔음
서브스크라이버 :
신문 구독하는 사람. 신문이 나오면 구독한 사람에게 신문이 배달됨.

어느 버퍼를 듣겠다고 한 노드가 있는 것. (뉴스에 대한 버퍼를 구독)

메세지로 핸들을 할 수 있음.