Software Architecture

Distributed Architecture

Types of Distributed Architecture Style

- Client-server
- Multitier
- Proxy
- Dispatcher (Load Balancer)
- P2P
- Broker
- Service-oriented architecture
- Microservice architecture

Proxy Architecture Style

Eunmi Choi
Kookmin University

클라이언트 단에 위치하여 진짜를 invocation하는 것.
외부의 서비스를 직접 부르는 것과 동일한 서비스를 할 수 있음.

Proxy: Synopsis

- The proxy forces method calls to an object to occur indirectly through a proxy object that acts as a surrogate for the other object, delegating method calls to that object
- Proxy objects generally share a common interface or superclass with the service-providing object

Proxy: Context

- Transparent management of another object's services is the basic reason for using a proxy.

transparancy : 투명성 . 시스템 투명성은 우리가 인보케이션 하는 것이 끝까지 같은 결과를 내는 것에 대해 신뢰할 수 있음.

- Different types of service management
  - A proxy makes a method that can take a long time to complete appear to return immediately

메소드를 부르면 일을 하는데 시간이 걸려서 작업을 하고 부름.
자기 것을 하면 오래걸리는데 다른 애가 하면 다른 것을 하다가 받을 수 있음.
비동기적인 일을 할 수 있음.

  - A proxy creates the illusion that an object that exists on a different machine is an ordinary local object (Remote Proxy)

원격 프록시 : 자기 단에서 돌리는 것이 아닌데도 큰 서비스를 돌리는 것과 같은 환상을 줌.

  - A proxy controls access to a service-providing object (Access Proxy)

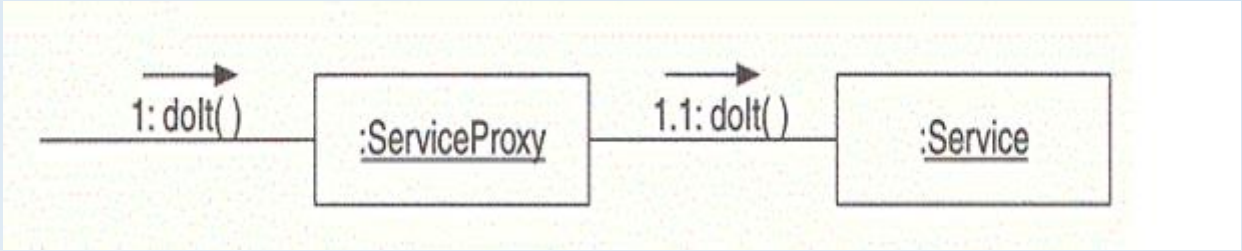
외부 서비스에 접속하려면 절차가 복잡함. 접속을 하려면 진의 여부, 권한이 있는지에 따라 접근 여부가 결정됨. 이런 것을 하려면 오래 걸리니 자기가 적절한 사용자라 느끼면 외부에 접근할 수 있는 프록시를 가져 가능하게 함.

  - A proxy creates the illusion that a service object exists before it actually does (Virtual Proxy)

프록시는 간략한 테이블 정도를 넣어놓고 이걸로만 작업을 함. 서비스가 완전히 되면 서버에 그 결과를 반영시킴.
그러면 실제로 돌아가는 서비스가 나중에 반영되도록 함.(8분30초쯤)
클라이언트와 서버가각자의 일을 할 수 있음. 부담없이.

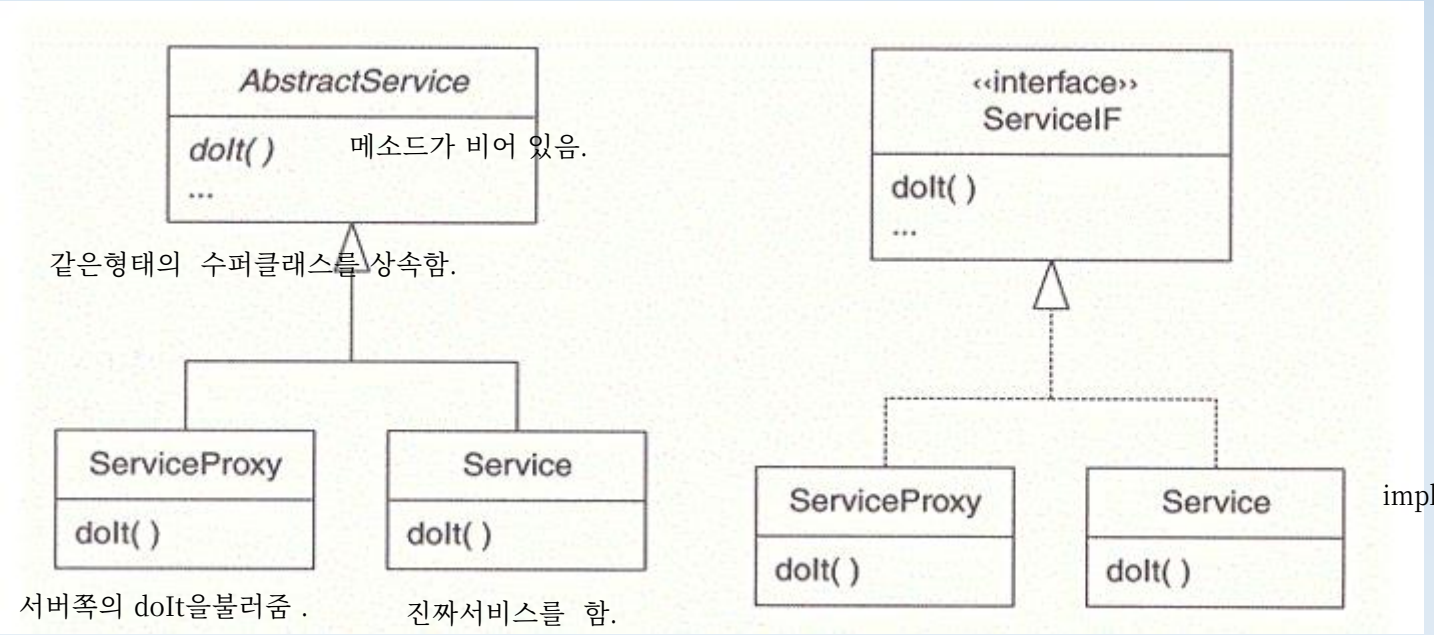
Proxy: Context

- Method calls through a proxy



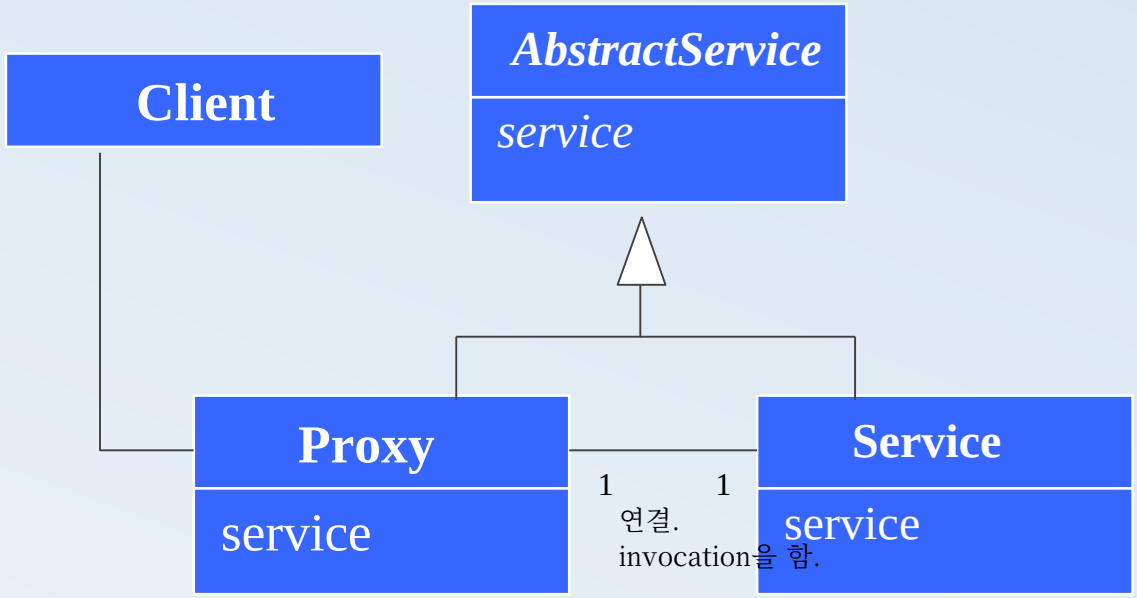
Proxy: Structure 1

- Proxy class diagram



Proxy: Structure 2

앞에 두 그림과 이 그림 세개 모두 프록시를 나타내는 그림임!
세가지 다 가능한 그림 표현.



프록시가 클라이언트를 연결

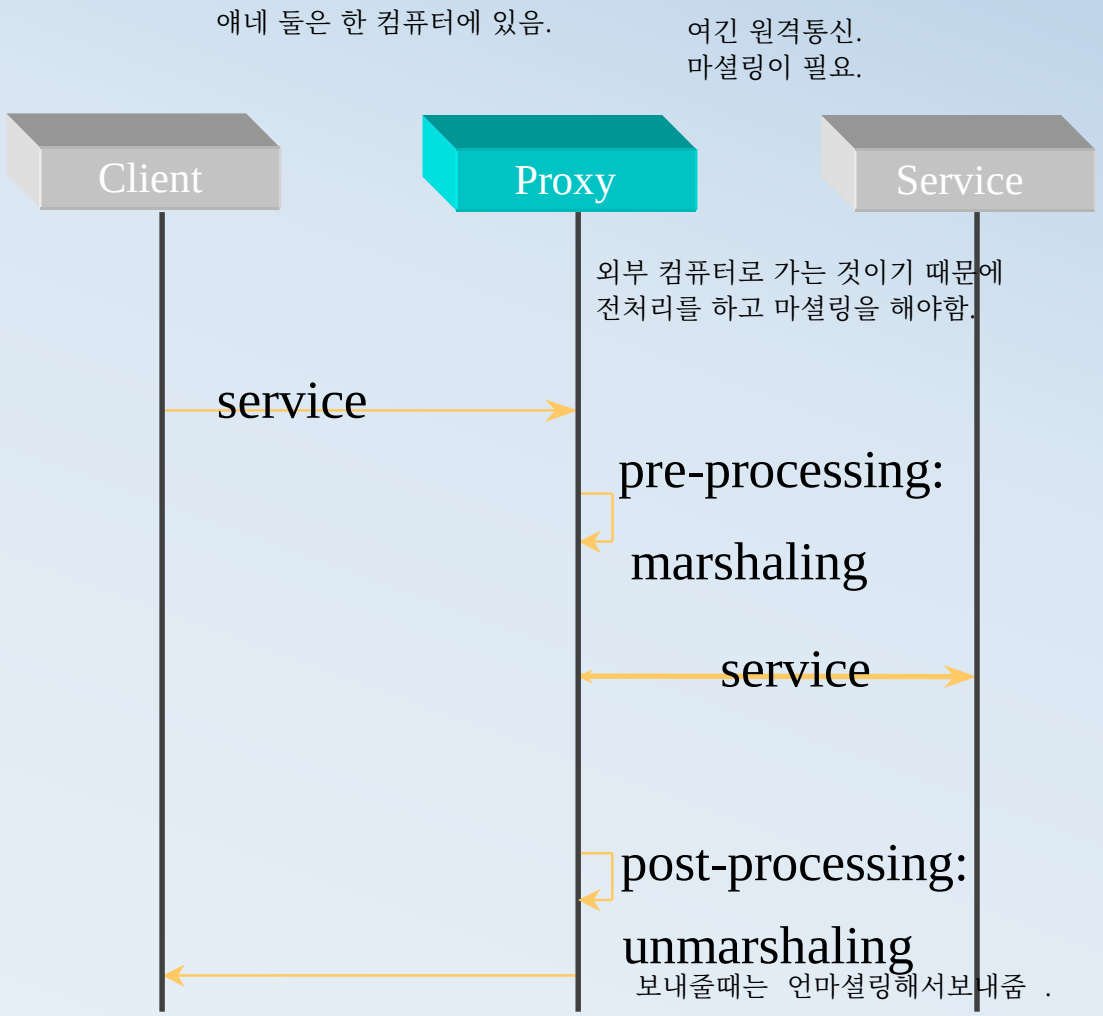
- Solution

오브젝트에 대한 플레이스 홀더의 역을 함.
이 위치에 대한 서비스를 잡아놓음(placeholder)
연결점을 만들어놓음.

  - Provide a placeholder for the object through which clients can access it
  - A *Service* implements the object which is not directly accessible.
  - A *Proxy* represents the Service and ensures the correct access to it. The Proxy offers the same interface as the Service.
  - *Clients* use the Proxy to get access to the Service.

indirect하게 연결할 수 있음.

Dynamics


얘네 둘은 한 컴퓨터에 있음.
여긴 원격통신.
마셜링이 필요.
Client
Proxy
Service
외부 컴퓨터로 가는 것이기 때문에
전처리를 하고 마셜링을 해야함.
service
pre-processing:
marshaling
service
post-processing:
unmarshaling
보내줄때는 언마셜링해서보내줌 .

Benefits and Liabilities

- Benefits
  - Access control to originals.
  - Memory savings.
  - Performance gaining (cache proxy).
  - Separation of housekeeping and functionality.

오리지널한 오브젝을 접근할 수 있는 권한이 매우 좋음

클라이언트 쪽에서 헤비한 서비스를 운영하지 않는데도 결과적으로 메모리 세이빙이 클라이언트에서 돌아감

다른 쪽에서 일을 하고 프록시에는 캐시만 만듦.
거기에 따른 비직접적인 단계를 넣어 전체적인 퍼포먼스가 좋아짐.

집안 관리 - 서버쪽에서 관리를 하는 관리자 쪽과 유즈만 하는 클라이언트.
하우스 키핑(관리)하는 쪽과 사용하는 쪽을 분리함.

- Liabilities
  - Potential overkill, if proxies include overly sophisticated functionality.
  - Level of indirection.

고려할 점 >
프록시는 오버레이를 함. 따라서 요청을 했는데 끊기면 어디서 문제가 생기는지 잘 모르겠음.
어디서 끊어진건지 추적하기가 어려움.(인다이렉션 단계가 많아질 수록 심해짐)