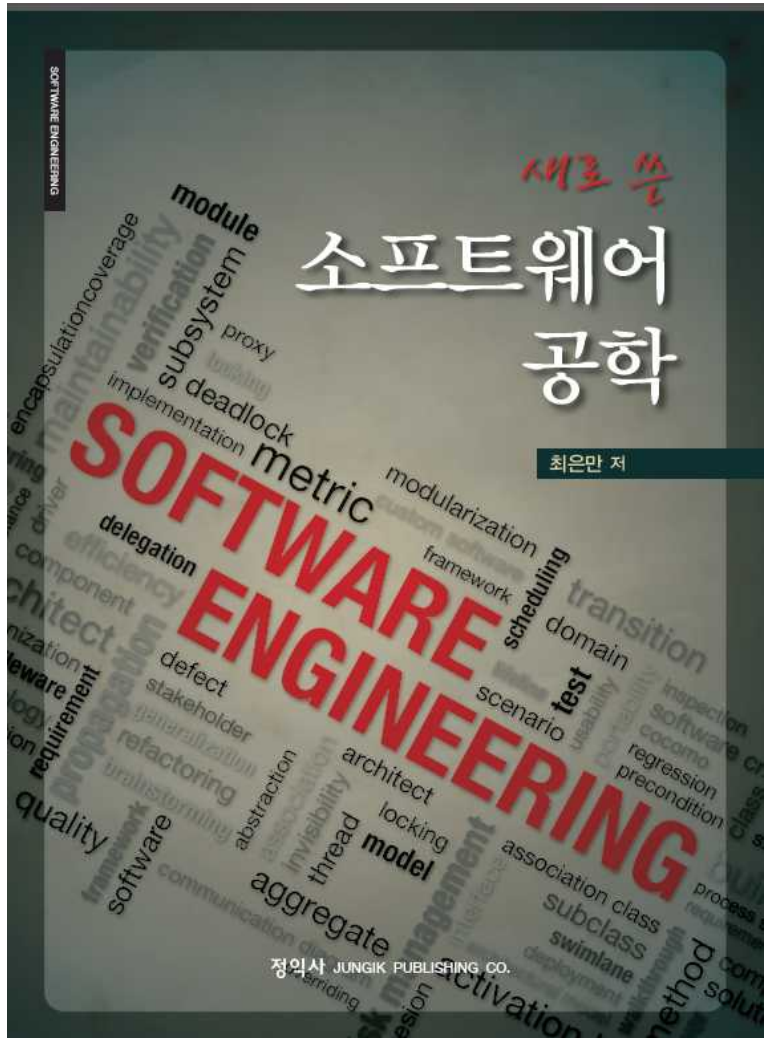
소프트웨어 공학 개론

강의 8: 설계원리와 아키텍처

최은만
동국대학교 컴퓨터공학과

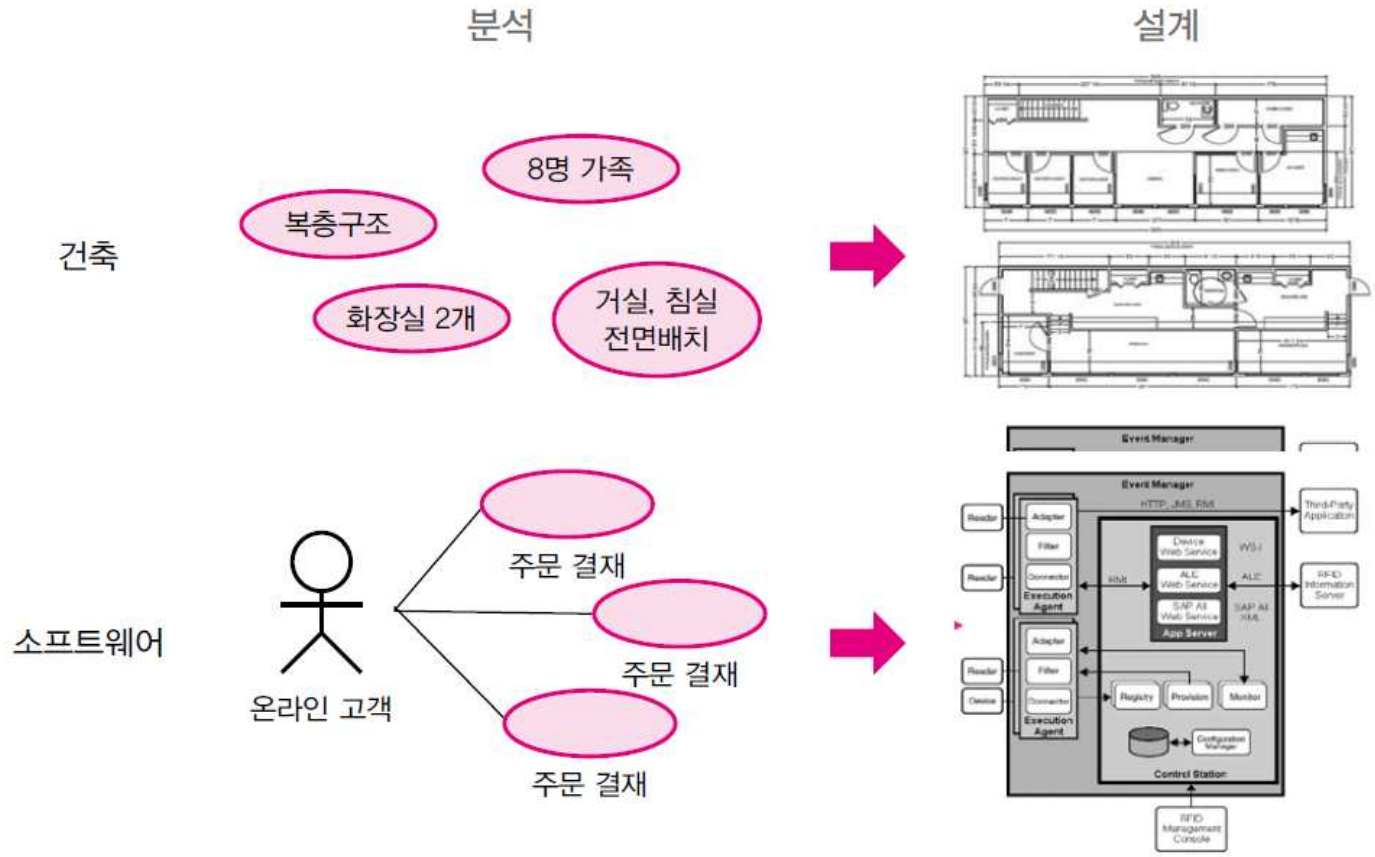


학습목표

- 아키텍처 설계란?
- 설계 원리
- 아키텍처 설계 과정
- 아키텍처 스타일
- 미들웨어 아키텍처
- 설계 문서화

요구 분석에서 설계로

- 요구 분석 작업을 통하여 무엇을 개발할 것인가를 결정한 후에는 도메인 영역의 문제에 집중하여 모델링한다.



설계

- 대규모 소프트웨어를 설계하기 위한 작업은 다음과 같이 분류한다.
  - 소프트웨어 아키텍처 설계
  - 인터페이스 설계
  - 자료 저장소 설계
  - 모듈 설계
  - 사용자 인터페이스 설계

6.1 아키텍처 설계란?

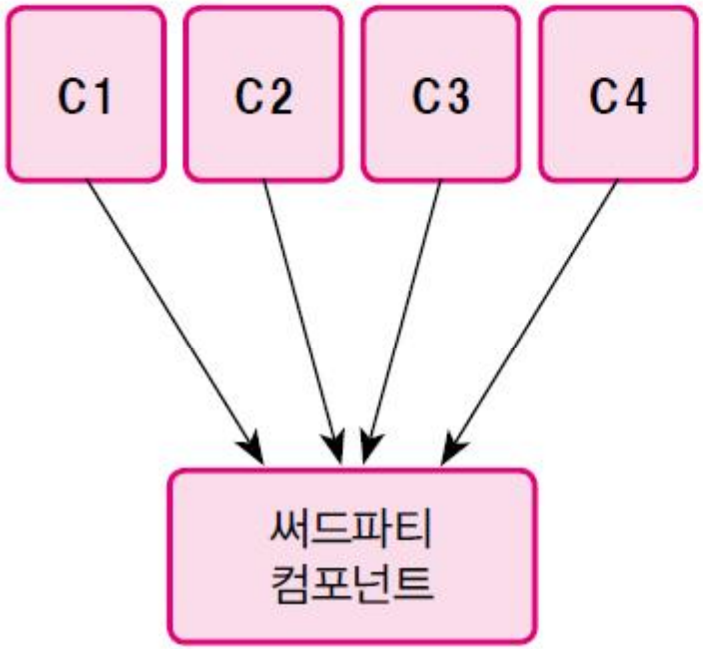
- 아키텍처 설계의 정의
  - 소프트웨어 아키텍처란 주요 컴포넌트 사이의 인터페이스와 인터랙션을 포함한 시스템 구조의 설계 유형을 말한다. 아키텍처 설계는 개발 중인 시스템에 대한 아키텍처를 정하는 의사 결정 과정이다.

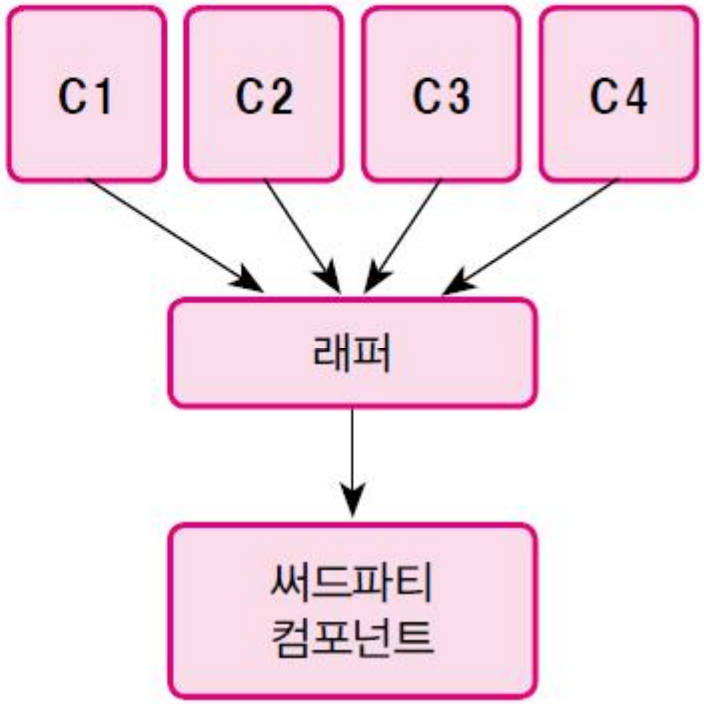
- 컴포넌트
  - 컴포넌트란 명백한 역할을 가지고 있으며 독립적으로 존재할 수 있는 시스템의 부분이다. 같은 기능을 가진 다른 컴포넌트로 대체 시킬 수 있다.

- 모듈(Module)
  - 모듈이란 프로그래밍 언어의 문법 구조에서 정의된 컴포넌트를 말한다.
    - 예) 메소드, 클래스, 패키지는 Java프로그램의 모듈이다.
    - 예) C프로그래밍 언어에서의 모듈의 파일과 함수이다.

아키텍처 설계의 중요성

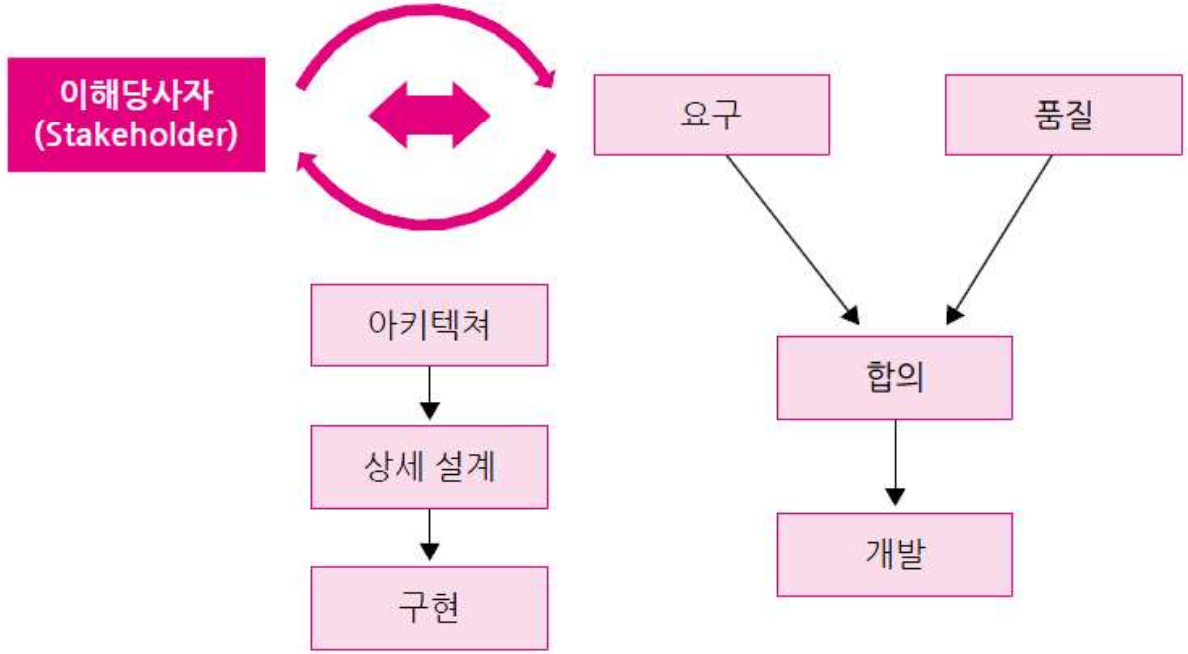
- 아키텍처 설계의 비교





아키텍처와 비기능적 요구(1)

- 비기능적 요구 균형
  - 기술적 제약
  - 비즈니스 제약
  - 품질 제약
- 아키텍처 설계와 품질



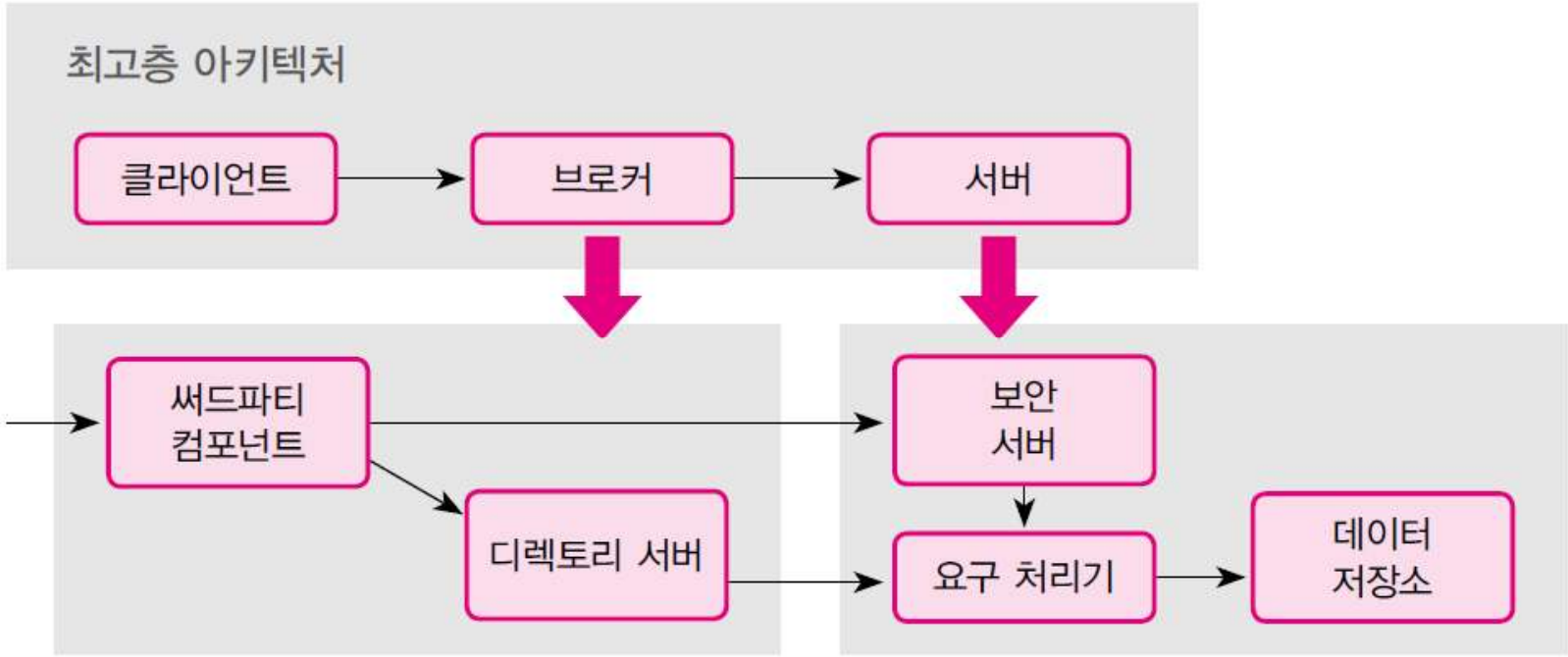
아키텍처와 비기능적 요구(2)

- **품질 제약**

품질	속성	설계목표
성능	처리량(throughput)	시스템이 단위 시간 당 얼마나 많은 작업을 하여야 하는가? 단위는 초당 처리한 트랜잭션 수(transaction per second) 또는 메시지 수 (massage processed per second) 예를 들면 온라인 뱅킹 시스템은 1,000tps의 성능을 보장하여야 한다. 주어진 시간 동안(하루 또는 한 시간)의 평균 처리량을 제시할 수도 있다.
	반응시간(response time)	사용자의 작업 요청 또는 입력 후 얼마나 빨리 시스템이 반응하는가? 보장 반응시간과 평균 반응시간 두 가지로 정의한다. 전자는 모든 요구가 정해진 시간 안에 반응하여야 하고 후자는 최악으로 지연되는 것을 허용하는 방법이다. 두 가지를 절충하여"95%"의 요구는 4초 이내에 반응하여야 하나 15초 이상이 되어서는 안된다 : 라고 목표를 설정할 수도 있다.
	메모리	시스템이 실행되기 위하여 요구되는 기억공간은 얼마인가? 임베디드 시스템의 경우 메모리를 무한정 소모 할 수 없다.
확장성	동시접속자 수	확장성이란 사용자수의 증대에 유연하게 대처할 수 있는 성질이다. 동시접속자 수는 확장성의 목표로 사용될 수 있다. 예를 들어 "1만 명의 사용자가 동시에 접속하여야 한다."
	데이터 크기	메시지의 크기가 늘어난다고 해도 애플리케이션의 아키텍처가 잘 견디는가? 어느 정도까지 큰 무리 없이 처리할 수 있는지를 나태냄
변경용이성	수정비용	새로운 기능적 요구나 비기능적 요구를 충족하기 위하여 얼마나 쉽게 애플리케이션을 수정할 수 있는가?
보안	기밀성	정확히 계량화 하기는 어렵지만 인증, 권한부여, 암호화가 잘 되어야 한다.
가용성	가동률	정해진 시간 동안(또는 영업시간 동안)에 시스템이 사용될 수 있도록 가동중인 비율. 대부분의 IT애플리케이션은 항상 가동되어야 하므로 100%의 가동률을 요구한다.
통합	호환성	데이터의 호환성을 제공하거나 다른 시스템과의 인터페이스를 제공하는 것을 의미한다.

아키텍처의 표현

- 아키텍처의 계층적 분할



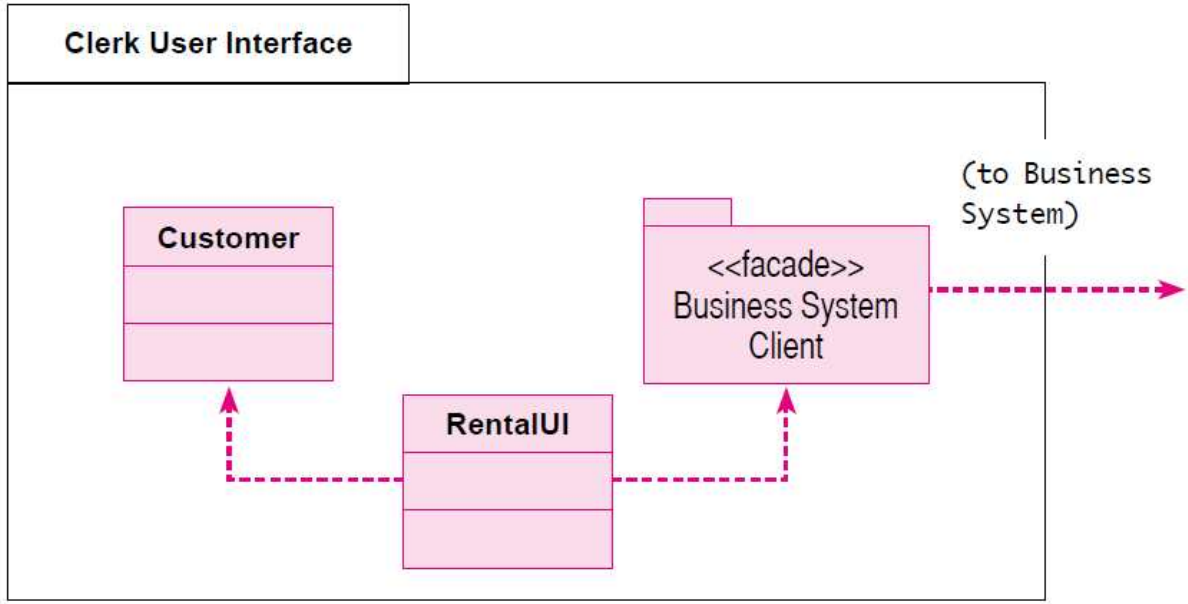
패키지 다이어그램

- UML에서 서브시스템을 표현할 때 도입되는 개념
  - 클래스를 의미 있는 관련된 그룹으로 구성하는 메커니즘
  - Java의 Package 선언으로 매핑
  - 패키지는 중첩될 수 있음
  - 복잡한 시스템을 서브시스템으로 나누어 적절한 컨트롤 가능
  - 소프트웨어 구조를 표현하는데 적합
  - 서브시스템으로 분할하면 객체 사이의 의존성을 최소화할 수 있어 솔루션 도메인의 복잡성을 줄일 수 있음
  - Façade 패턴

패키지 다이어그램

- 패키지 다이어그램에서의 의존 관계

- 패키지로 정의한 서브 시스템

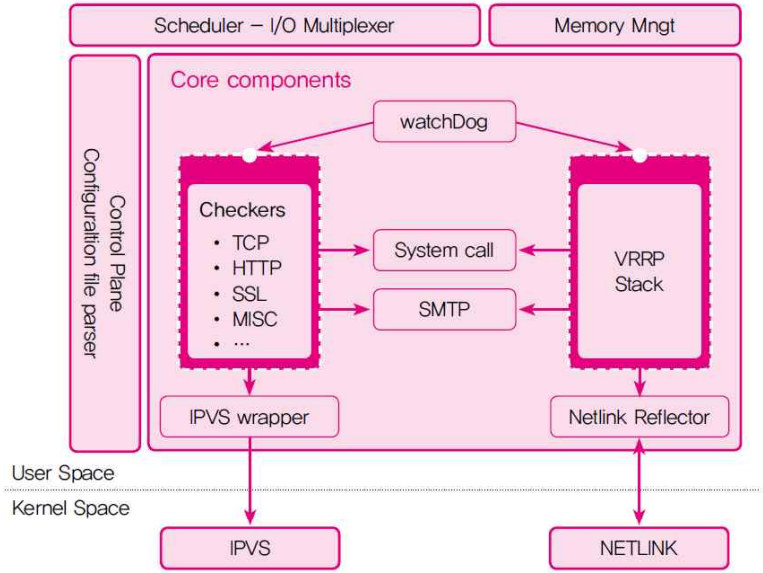


6.2 설계 원리(1)

- 시스템을 개발하는 조건이나 운용될 환경 조건의 제약 안에서 가능한 여러 설계 중에서 최적의 설계 안을 발견 하는 것
  - 설계를 평가하기 위한 특성과 기준을 명시하여야 한다. (정량적)
  - 효율성(Efficiency)
    - 시스템이 사용하는 자원이 적정하고 효과적임을 의미한다.
  - 단순성(Simplicity)
    - 이해하기 쉬운 설계를 작성하는 것

- 소프트웨어 설계의 중심이 되는 원리
  - 추상화(Abstraction)
  - 정보은닉(Information hiding)
  - 단계적 분해(Stepwise refinement)
  - 모듈화(Modularization)

6.2 설계 원리(2)

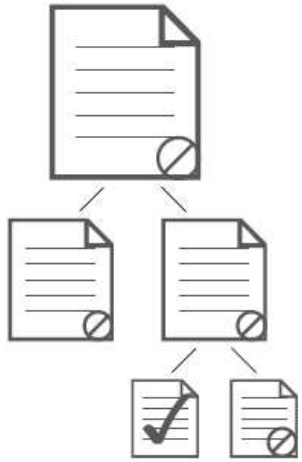


단순성	효율성	분할, 계층화	추상화	모듈화
복잡한 여러 가지 요소를 교통 정리하여 단순화 하거나 복잡함을 최소화 한다.	사용하는 자원이 적정하고 효과적이도록 한다.	다루기 쉬운 덩어리로 분리하여 계층화 한다.	자세한 부분에 좌우되지 않게 컴포넌트를 정의한다.	각 모듈이 외부와의 결합이 낮고 내부 요소가 응집되도록 한다.

단계적 분할

- 단계적 분해
  1) 문제를 기본 단위로 나눈다.
  2) 독립된 문제로 구별한다.
  3) 구분된 문제의 자세한 내용은 가능한 한 뒤로 미룬다.
  4) 구체화 작업이 계속 점증적으로 일어난다는 것을 보인다.

- 문제의 분할 개념



추상화(Abstraction)

- 정의

  필요한 부분만을 표현할 수 있고 불필요한 부분을 제거하여 간결하고 쉽게 만드는 작업.

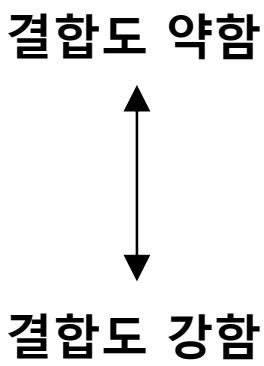
- 추상화 방법
  - 기능 추상화
  - 자료 추상화
  - 제어 추상화

모듈화(1)

- 설계 과정에 독립된 모듈들로 분할하기 위하여 모듈 선택에 사용하는 기준이 있다. 기능 추상화 기법을 사용하는 시스템에서는 결합(coupling)과 응집(cohesion)이라는 두 가지 관점의 판단 기준이 적용된다.

모듈화(2)

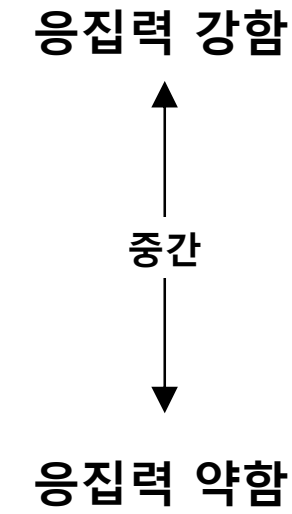
- 모듈 간의 결합(coupling)
  (1) 자료 결합(data coupling)
  (2) 스탬프 결합(stamp coupling)
  (3) 제어 결합(control coupling)
  (4) 공통 결합(common coupling)
  (5) 내용 결합(content coupling)



	인터페이스 복잡도	연결 형태	커뮤니케이션 내용
낮음	단순 명확	이름으로 모듈에 연결	데이터 제어정보
높음	복잡하고 불투명	내부 요소로 연결	복합

모듈화(3)

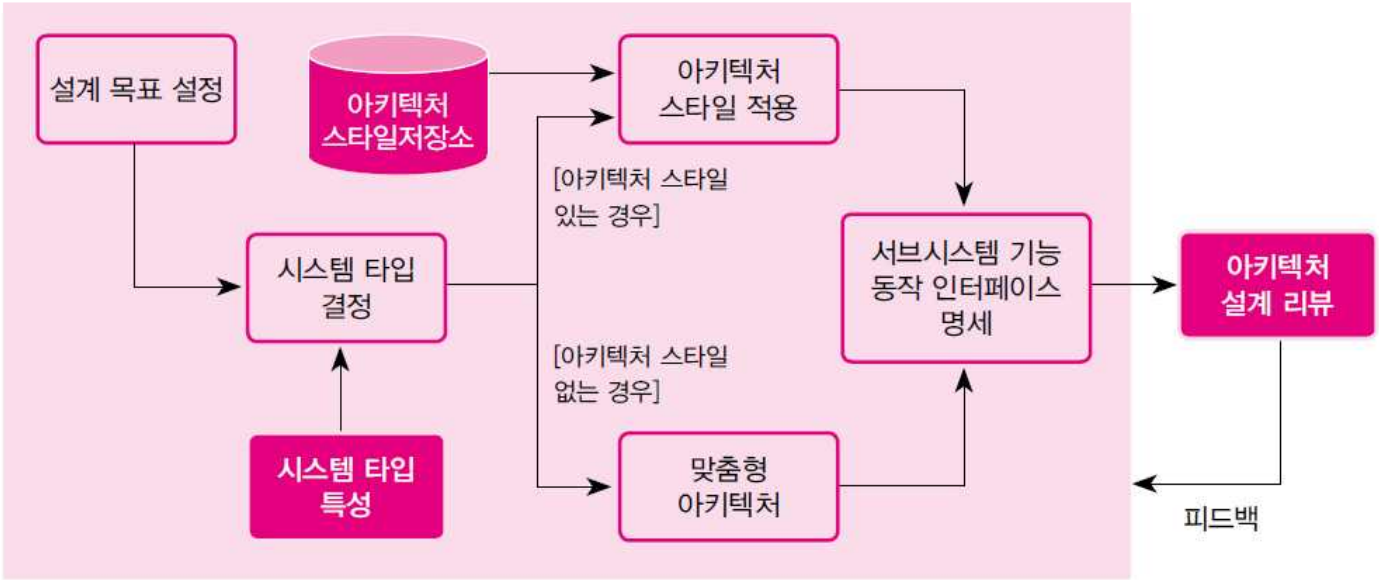
- 모듈의 응집(cohesion)
  - (1) 기능적 응집(functional cohesion)
  - (2) 순차적 응집(sequential cohesion)
  - (3) 교환적 응집(communication cohesion)
  - (4) 절차적 응집(procedural cohesion)
  - (5) 시간적 응집(temporal cohesion)
  - (6) 논리적 응집(logical cohesion)
  - (7) 우연적 응집(coincidental cohesion)



모듈 기능을 정의한 문장	응집력
복문, 쉼표 하나 이상의 동사	순차적이나 교환적 응집
시간과 관련된 단어(처음, 다음, 이후 등)	순차적이나 시간적 응집
동사 앞에 단일 특정 객체가 아닌 경우	논리적 응집
초기화, 모두 삭제	시간적 응집

6.3 아키텍처 설계과정

- 아키텍처 설계 프로세스
  - Step 1. 목표를 설정한다.
  - Step 2. 시스템의 타입을 결정한다.
  - Step 3. 아키텍처 스타일을 적용하거나 아키텍처 설계를 커스터마이즈 한다.
  - Step 4. 서브시스템의 기능, 인터페이스 인터랙션 동작을 작성한다.
  - Step 5. 아키텍처 설계를 검토한다.



설계 목표 설정

- 아키텍처 설계에서 고려해야 될 요구사항
  - 변경, 유지보수 용이성
  - 상용 컴포넌트의 사용
  - 시스템 성능
  - 신뢰성
  - 보안
  - 고장인내성
  - 복구

시스템의 타입

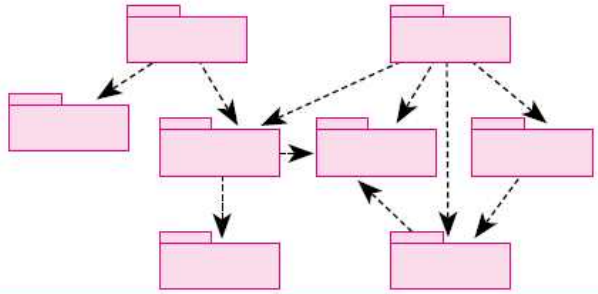
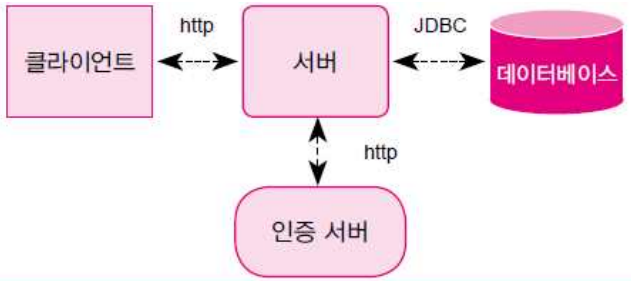
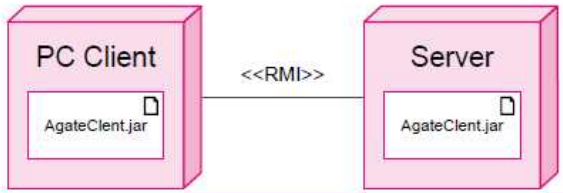
- 대화형 시스템
- 이벤트 중심 시스템
  - 상태에 의존적이며 반응 동작을 보인다.
- 변환 시스템
- 객체영속 시스템
  - 저장미디어를 숨기고 데이터베이스나 파일 시스템에 객체를 저장하고 검색할 수 있는 능력을 가진 시스템

아키텍처 표현

- 아키텍처 관점
  - 모듈
    - 시스템을 단위코드의 집합으로 보는 관점
  - 컴포넌트와 커넥션
    - 시스템을 컴포넌트라 부르는 런타임 개체의 집합으로 보는 관점
  - 배치
    - 단위 소프트웨어가 어떤 하드웨어 노드에 배치되는가에 관점

아키텍처 표현

- **아키텍처의 표현 관점**

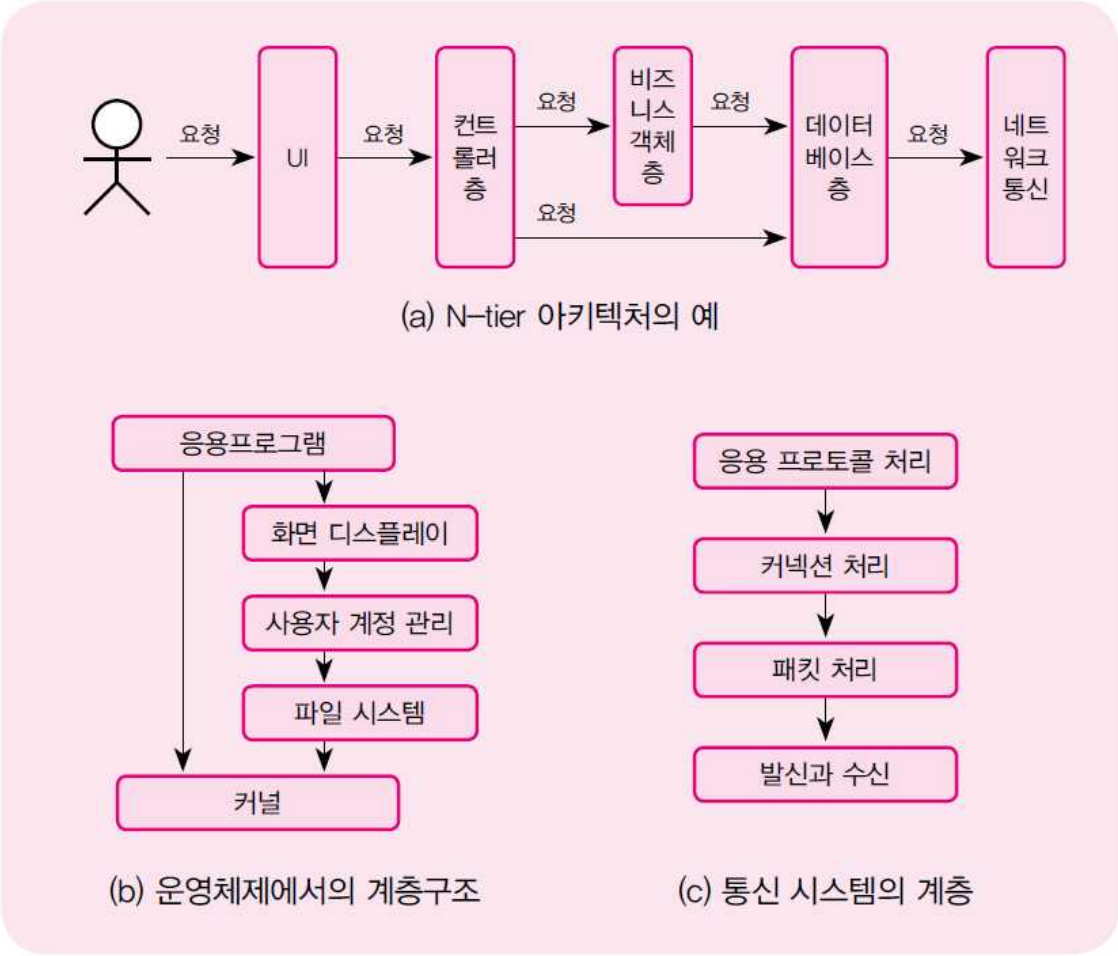
뷰	중심 관점	사례
모듈	코드 구조 – 의존, 상속, 부분 관계 등	
컴포넌트와 커넥터	런타임 구조 – 통신, 호출 관계	
배치	소프트웨어와 환경의 배치 구조	

6.4 아키텍처 스타일

- 일반적인 모양과 조화를 위한 스타일을 정하는 작업
  - 소프트웨어 아키텍처에 적용
  - 시스템분할, 전체 제어 흐름, 오류 처리 방침, 서브시스템 간의 통신 프로토콜 포함

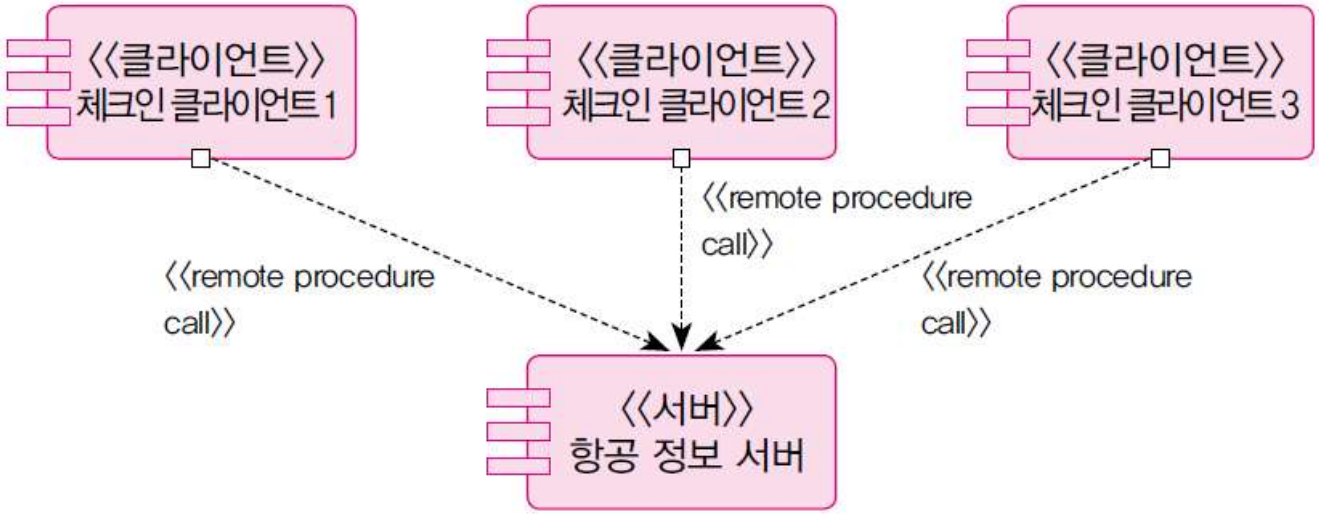
계층 구조 스타일

- 시스템을 계층으로 구성하는 방법



클라이언트 서버 스타일

- 클라이언트 서버(Client Server)
  - 서버와 여러 개의 클라이언트로 구성
  - 요청과 결과를 받기 위하여 동기화 되는 일을 제외하고는 모두 독립적이다.
  - 특정 서브시스템이 다른 서브시스템에 서비스를 제공하도록 지정할 때 유용하다.
  - 대부분의 웹 기반 애플리케이션과 파일 서버, 전송 프로토콜 포함



클라이언트 서버 스타일의 적용

트랜잭션 처리 스타일

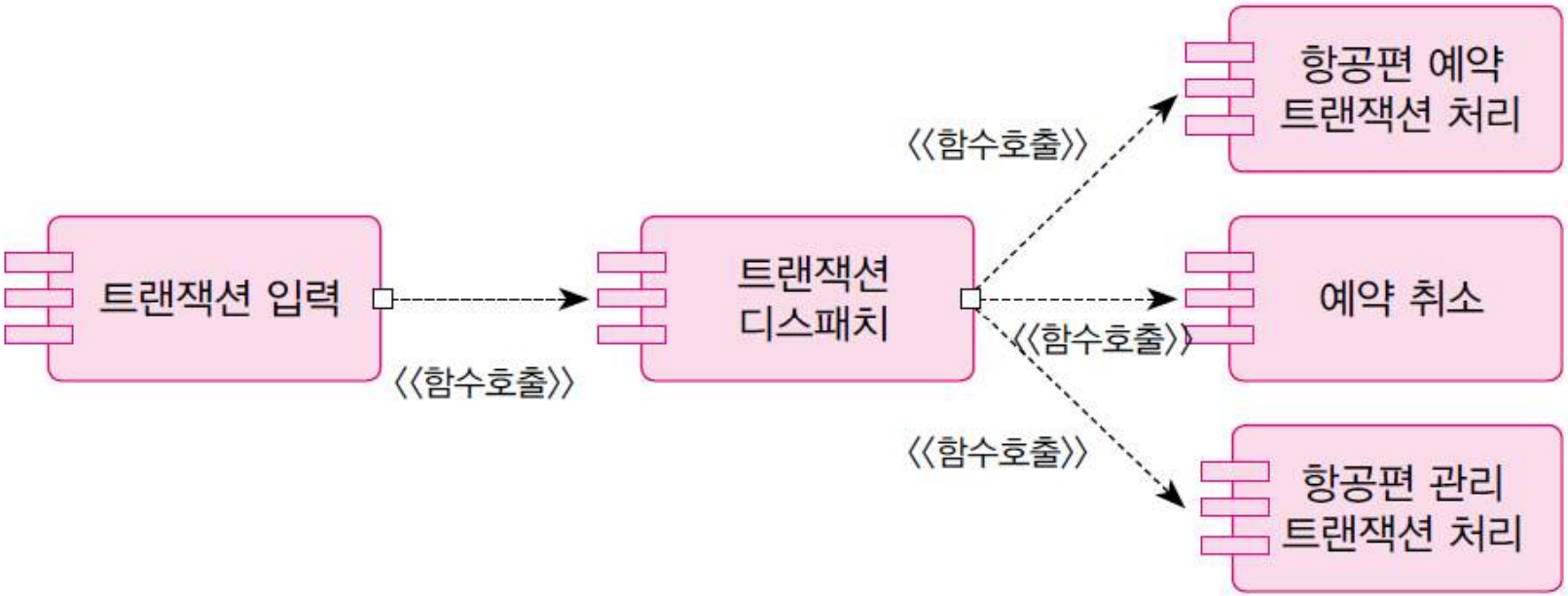
- 트랜잭션 처리 아키텍처는 입력을 하나씩 읽어 처리한다.
  입력은 시스템에 저장되어 있는 데이터를 조작하는 명령들,
  즉 트랜잭션이다.

- 트랜잭션을 어디서 처리하는지 결정하는 디스패처(Dispatcher)라고 하는 교통정리 컴포넌트가 필요하다.

- 디스패처는 프로시저 호출이나 메시지를 통하여 요청된 트랜잭션을 처리할 컴포넌트에 배치한다.

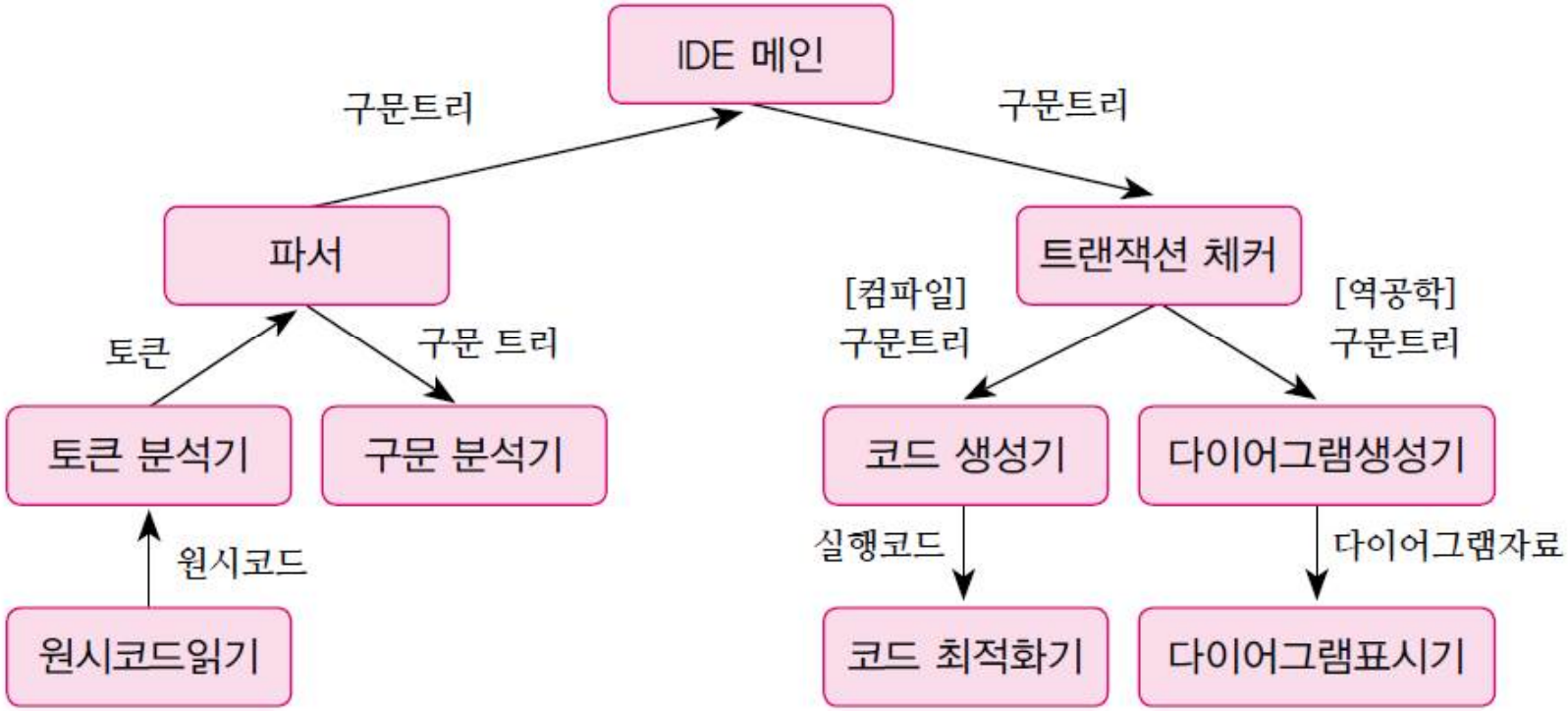
트랜잭션 스타일

- 트랜잭션 처리 스타일



트랜잭션 스타일

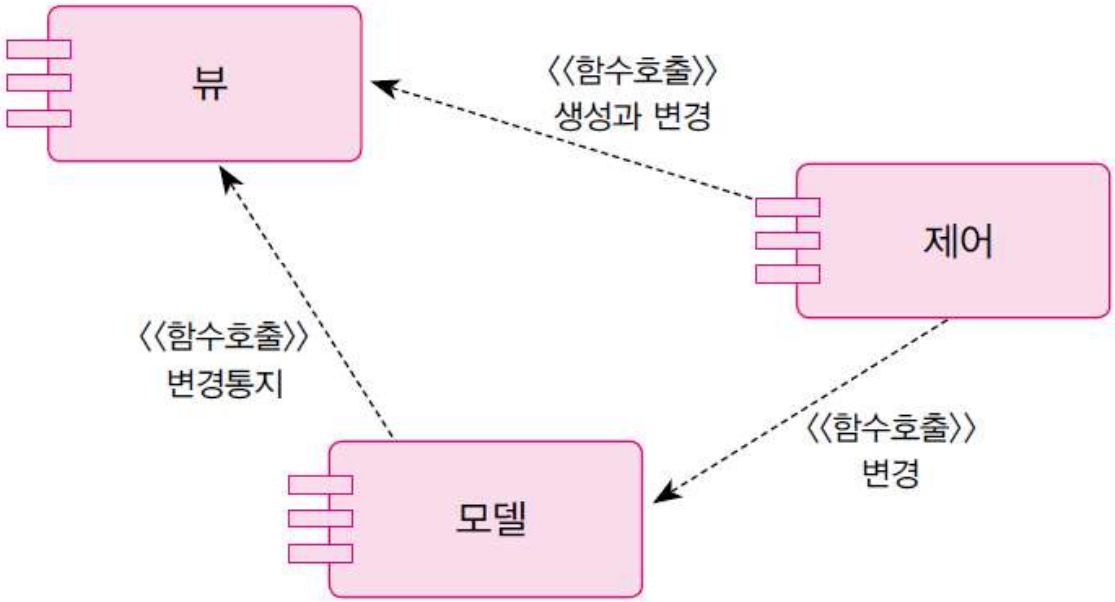
- 트랜잭션 처리 아키텍처 사례(IDE)



MVC 스타일

- **MVC(Model-View-Controller)**

  사용자 인터페이스를 시스템의 다른 부분과 분리하여 결합도를 낮추기 위한 아키텍처 스타일이다.

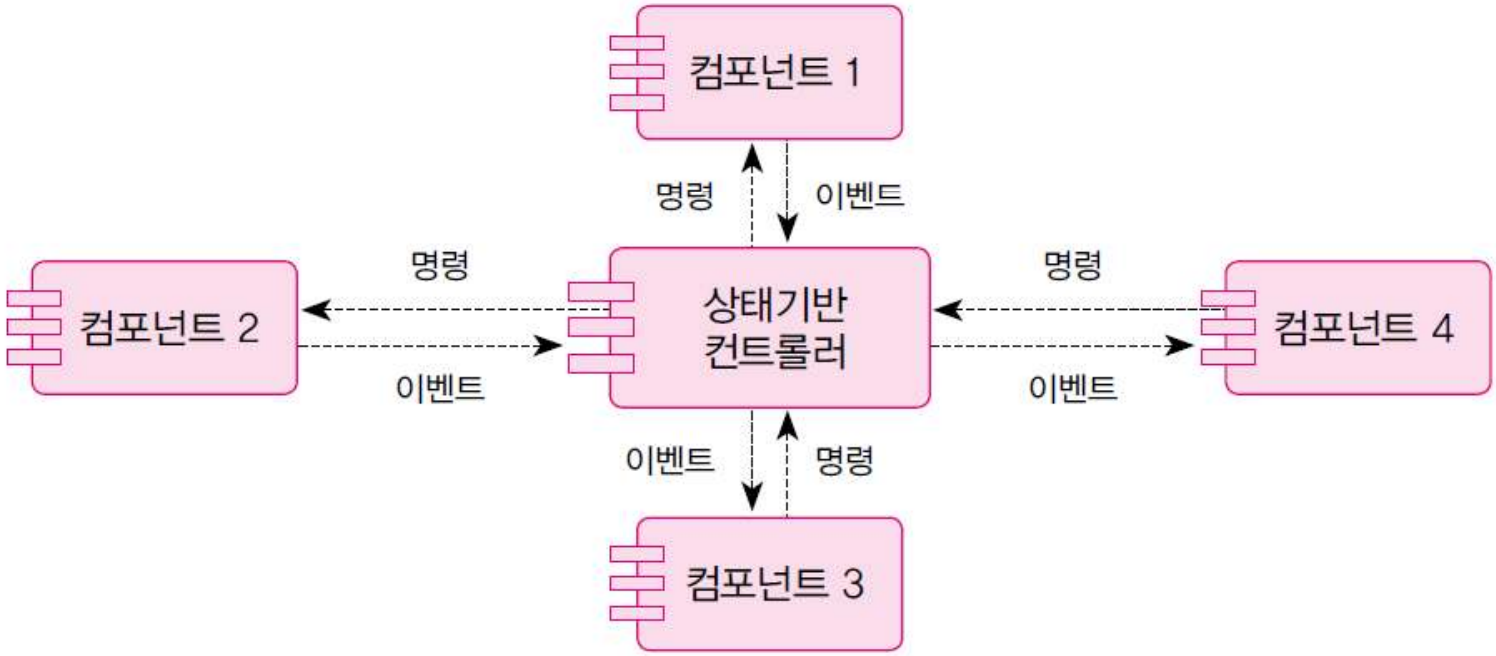


MVC 아키텍처 스타일

이벤트 중심 스타일

- 이벤트 중심 시스템 아키텍처는 상태 기반 컨트롤러와 제어 대상이 되는 여러 컴포넌트로 구성된다.

- 이벤트 중심 아키텍처



객체 영속 스타일

- 비즈니스 시스템은 데이터베이스에 객체를 저장하고 나중에 이를 검색할 필요가 있음
- 이런 시스템을 객체 영속 시스템 (Object Persistence System)이라고 함

6.5 미들웨어 아키텍처

- 소프트웨어 컴포넌트를 연결하기 위한 준비된 인프라 구조 제공
- 다양한 애플리케이션에 적용
- 일반적이며 환경에 따라 바꿀 수 있는 융통성을 가짐
  - 미들웨어는 애플리케이션의 여러 컴포넌트들을 연결하는 증명된 방법을 제공한다.
  - 미들웨어는 여러 컴포넌트를 1대1, 1대 다, 다대 다 등 여러가지 연결 형태로 연결하는데 유용하게 사용된다.
  - 사용자는 애플리케이션과 상호작용한다.
  - 미들웨어 역할을 알아야 하는 유일한 경우는 고장이다.

미들웨어 기술의 분류

BizTalk, TIBCO StaffWare, ActiveBPEL

Mule, WebSpere Message Broker

JEE, CCM, .NET

메시지 기반 미들웨어, 분산객체 시스템, SOAP

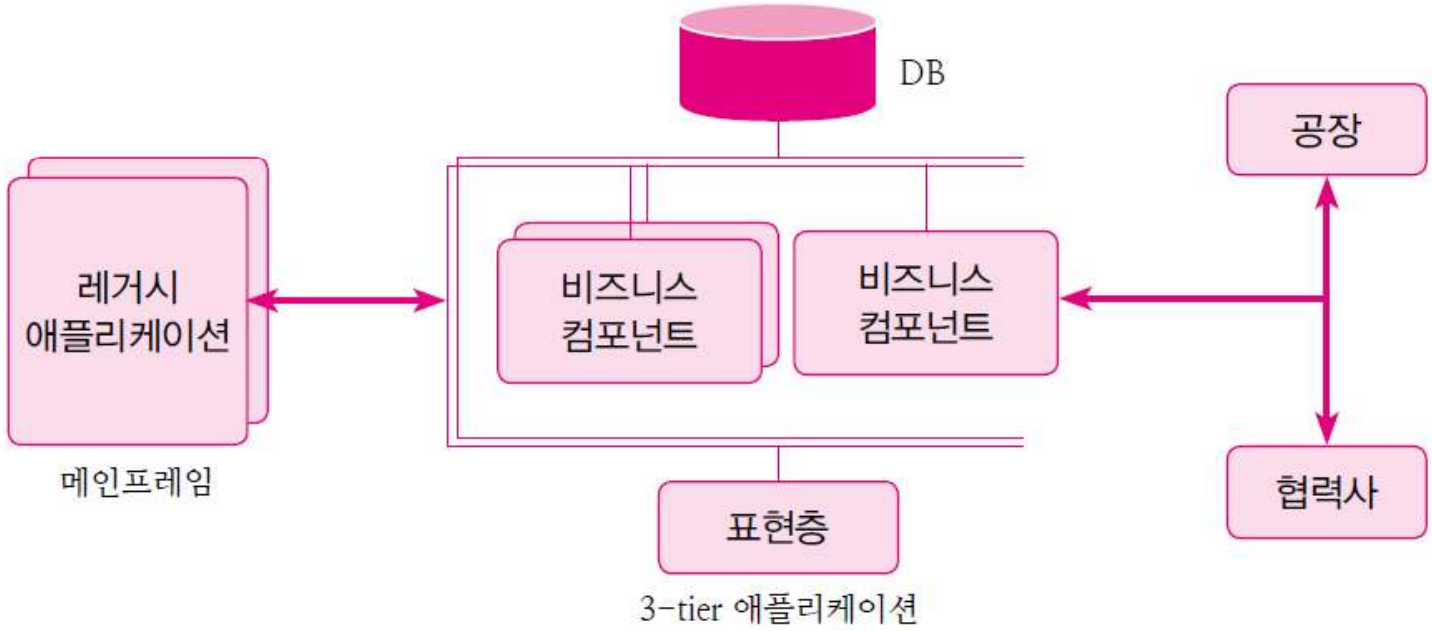
분산객체

- 분산 객체 기술은 미들웨어 기술 중의 하나이다. CORBA로 알려진 분산객체 기술은 1990년대부터 사용 되었다.
- CORBA에서는 서버 객체가 CORBA IDL (Interface Definition Language)을 이용하여 기술한 인터페이스를 지원한다.

```
module ServerExample{
    interface MyObject
    {
        String isAlive();
    };
}
```

메시지 중심 미들웨어(1)

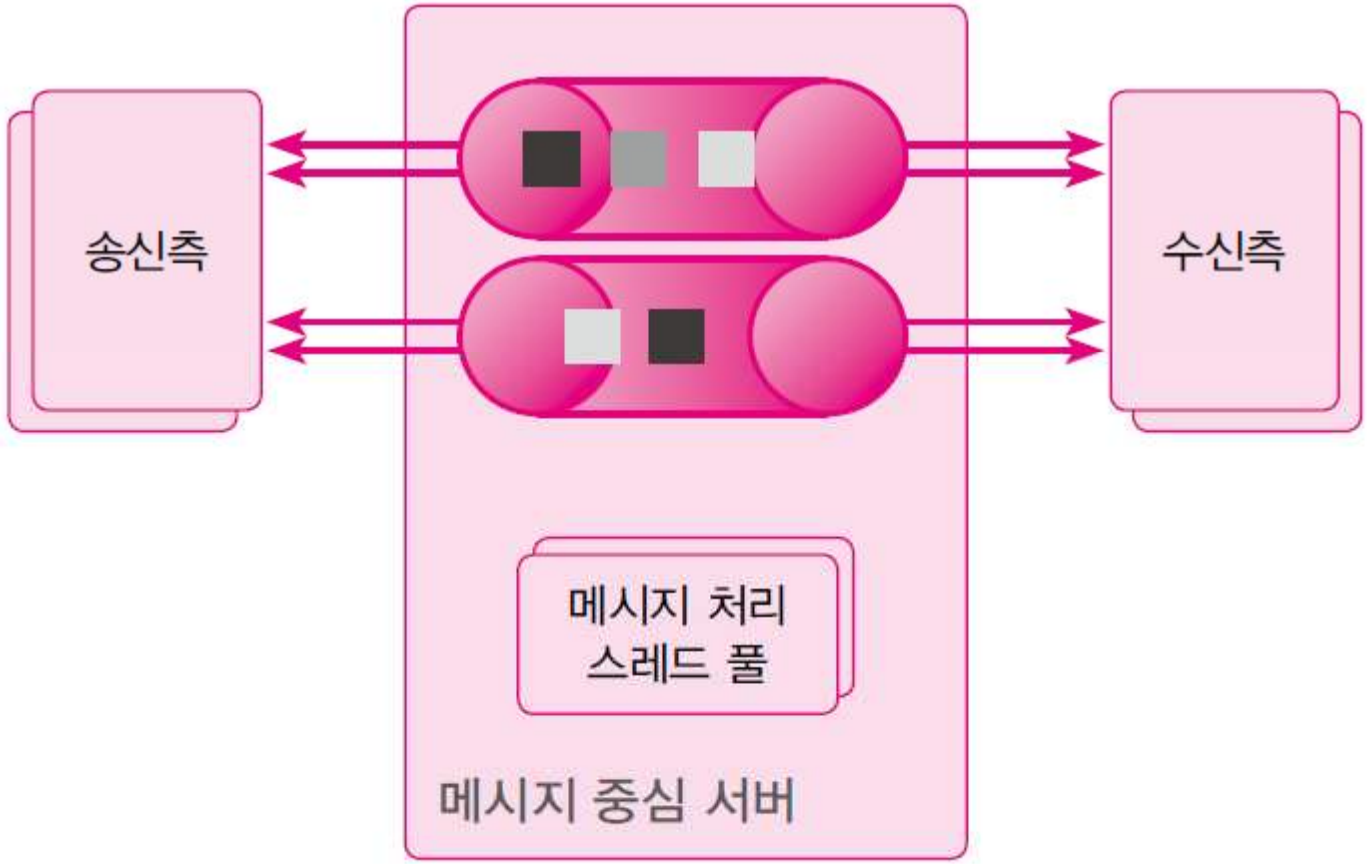
- 메시지 중심 미들웨어(Massage-Oriented Middleware) 는 대규모 엔터프라이즈 시스템을 구축할 때 중요한 기술이다. 서로 다른 독립적인 애플리케이션을 하나의 통합된 시스템으로 묶기 위한 접착제 같은 역할을 한다.



메시지를 통한 통합

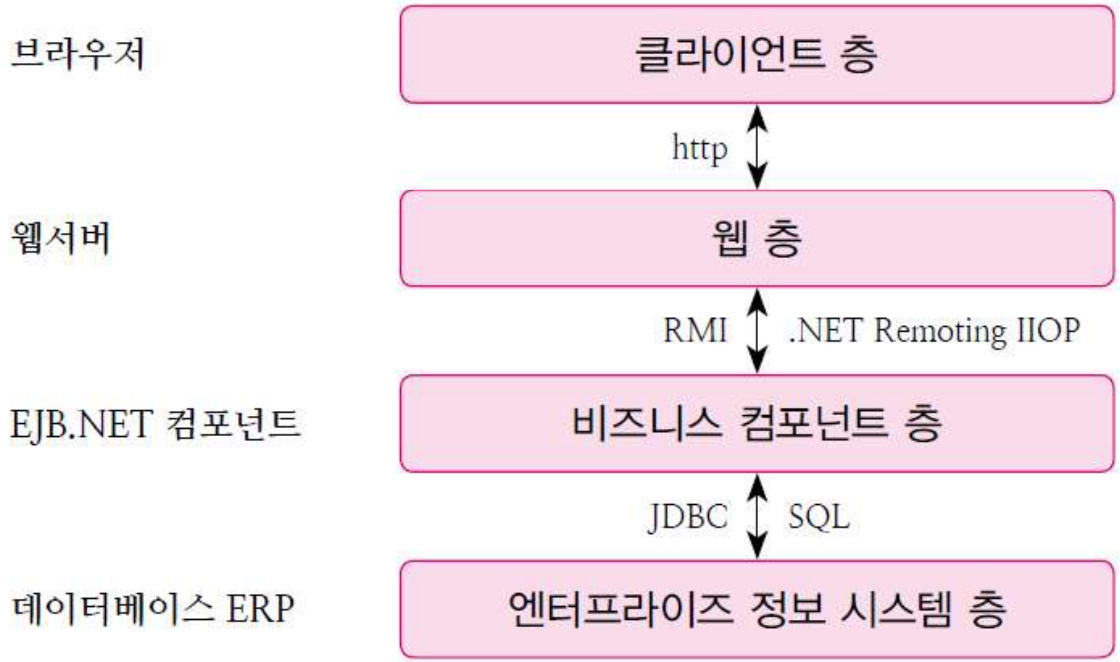
메시지 중심 미들웨어(2)

- 메시지 중심 서버



애플리케이션 서버

- N-tier 아키텍처의 중간층에 위치하면서 분산 통산, 보안, 트랜잭션, 영속성을 제공하는 컴포넌트 기반의 서버 기술
- 인터넷 기반의 애플리케이션을 구축하는 데 널리 이용



웹 매플리케이션을 위한 N-tier 아키텍처

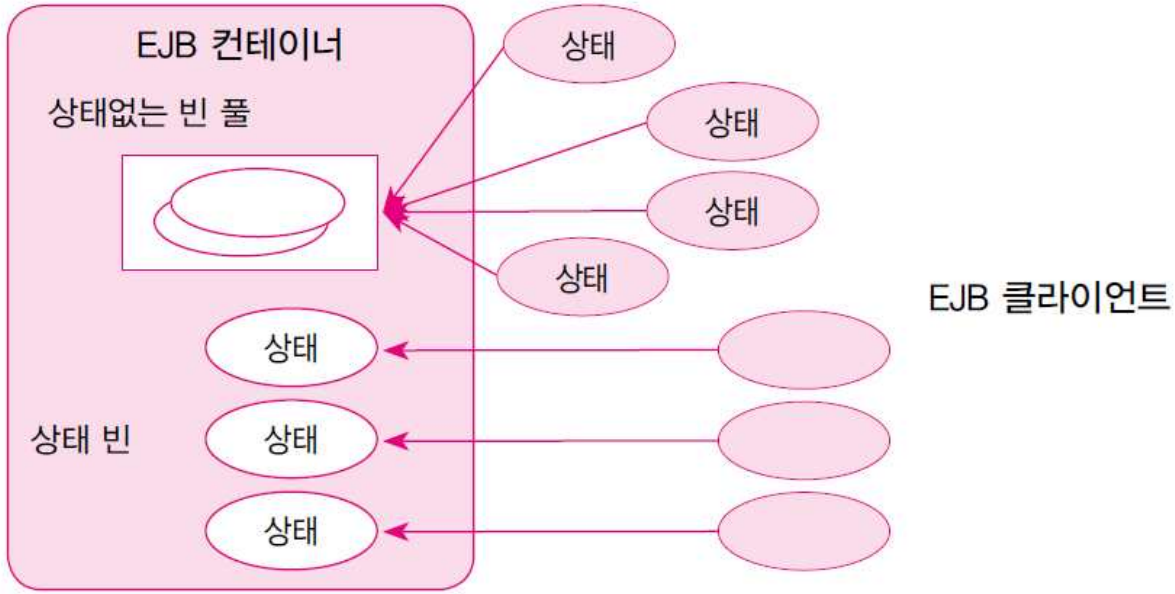
애플리케이션 서버

- **EJB 아키텍처**

JEE 애플리케이션 서버, EBJ 컨테이너 및 관련 서비스

애플리케이션 서버

- 세션 빈(session bean)
  - 일반적인 비즈니스 로직을 나타낸다.
  - 세션 빈은 두 가지 종류, 상태 없는(State-less)세션 빈과 상태(Stateful) 세션 빈이 있다.



상태 없는 빈과 상태 빈

6.6 설계 문서화(1)

- 아키텍처 문서를 작성하는 이유
  - 설계자나 설계 팀이 더 좋은 설계 결정을 내리는데 도움을 준다.
  - 다른 사람과 설계에 대하여 이야기를 할 수 있다.

- 설계문서는 다음 세가지 그룹이 의사 교환하는데 사용된다.
  - 설계를 구현할 사람 즉 프로그래머
  - 미래에 설계를 변경할 엔지니어
  - 설계된 시스템과 인터페이스 할 다른 시스템 또는 서브시스템을 개발하는 엔지니어

6.6 설계 문서화(2)

- 설계 문서는 일반적으로 다음과 같은 정보를 포함시킨다. 실제 적용하는 프로젝트에서 환경에 맞는 보다 자세하고 일관된 포맷을 사용하여야 할 것이다.
  - (1) 목적
  - (2) 우선순위
  - (3) 설계의 아웃라인
  - (4) 설계 이슈
  - (5) 설계 상세 사항
- 설계문서에서 제외할 사항
  - 훈련된 프로그래머나 설계가에게 당연한 것으로 여겨지는 정보를 문서화하는 것은 피한다.
  - 코드의 커멘트 안에 포함되어 있는 것이 더 좋을 내용은 설계 문서에서 피한다.
  - 코드에서 자동으로 추출될 내용, 예를 들면 public 메소드의 리스트는 설계문서에서 제외한다.

Questions?