



# **소프트웨어 공학 (Software Engineering)**

## **아키텍처 설계 (Architecture Design)**



## In this chapter ... (1/3)

아키텍처 설계(Architecture Design)

- 소프트웨어 아키텍처 무엇인가?
- 소프트웨어 설계 원리에는 어떤 것들이 있는가?
- 아키텍처를 설계하는 과정은?
- 아키텍처 스타일에는 어떤 것들이 있는가?
- 미들웨어 아키텍처 기술은 무엇인가?
- 아키텍처 설계 문서화 방법은?



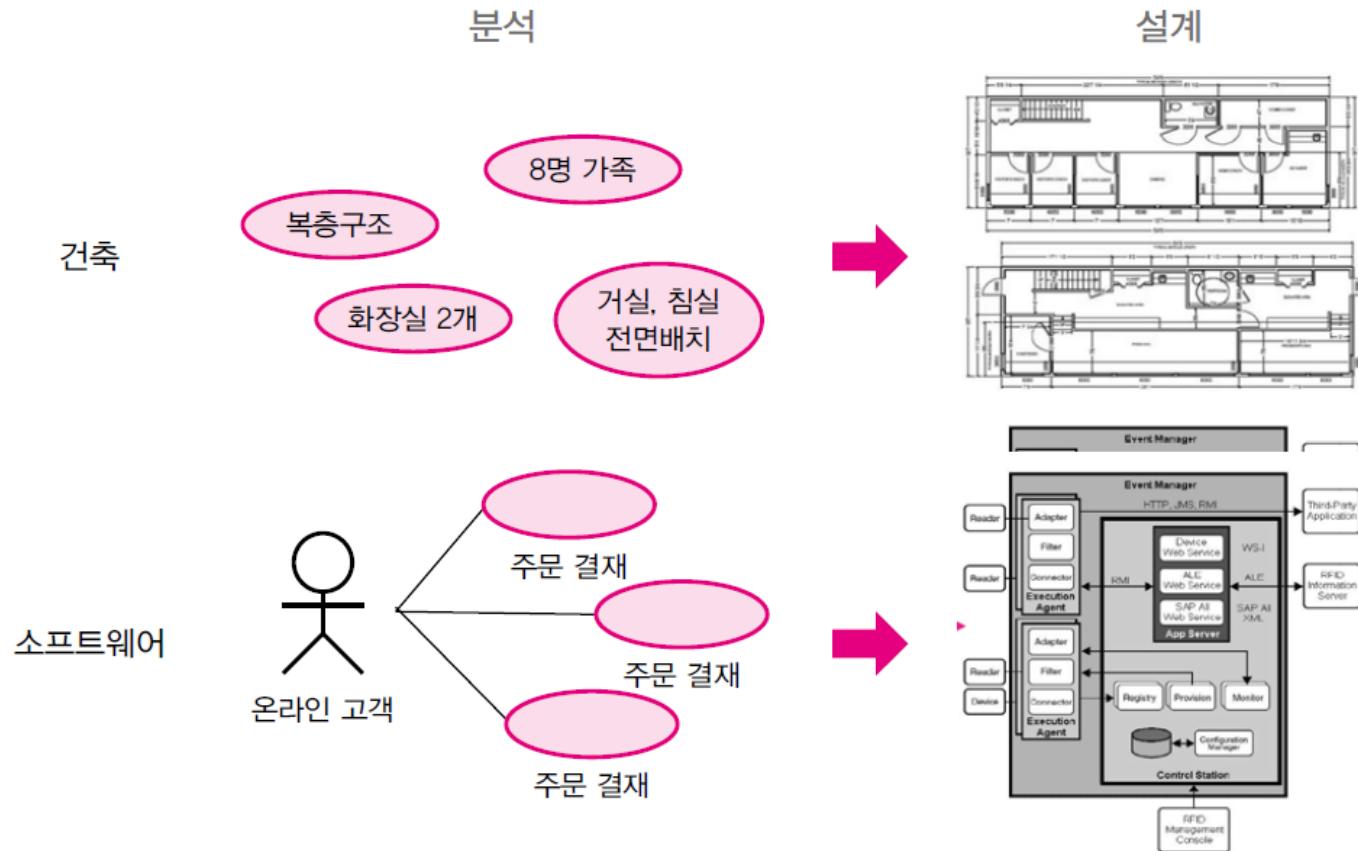
# In this chapter ... (2/3)

아키텍처 설계(Architecture Design)



## 요구 분석에서 설계로

- 요구 분석 작업을 통하여 무엇을 개발할 것인가를 결정한 후에는 도메인 영역의 문제에 집중하여 (구현을 위한) 모델링을 수행한다.





## 대규모 소프트웨어 설계를 위한 작업의 분류

- 소프트웨어 아키텍처 설계
- 인터페이스 설계
- 자료 저장소 설계
- 모듈 설계
- 사용자 인터페이스 설계





# In this chapter ...

아키텍처 설계(Architecture Design)



## 6.1 아키텍처 설계란?



## 6.2 설계 원리



## 6.3 아키텍처 설계 과정



## 6.4 아키텍처 스타일



## 6.5 미들웨어 아키텍처



## 6.6 설계 문서화

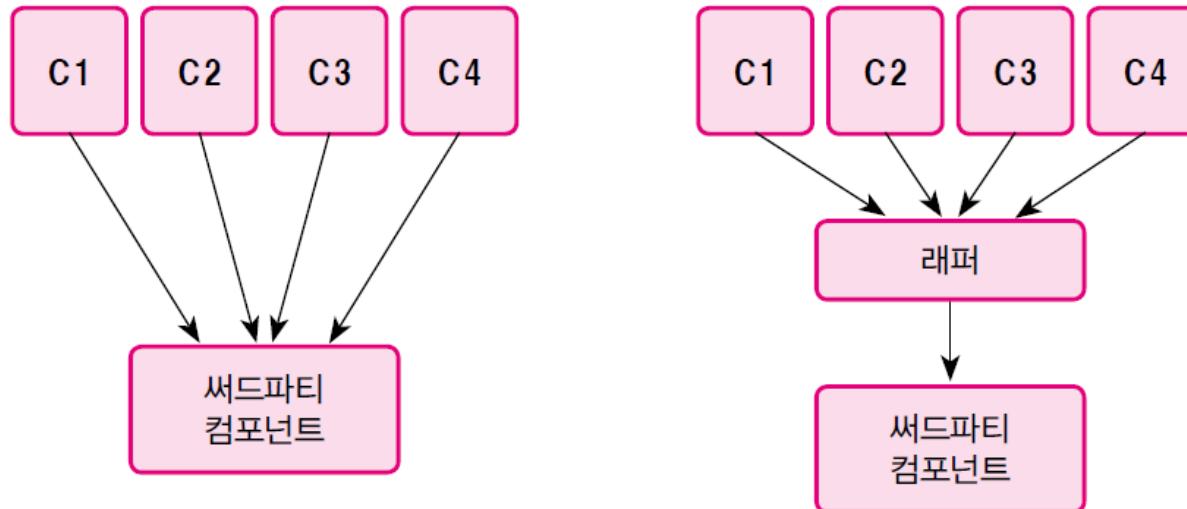


- **소프트웨어 아키텍처 설계는 빌딩의 아키텍처 설계와 유사함**
- **아키텍처 설계의 정의**
  - 소프트웨어 아키텍처란 주요 **컴포넌트** 사이의 인터페이스와 인터랙션을 포함한 시스템 구조의 설계 유형을 말한다.
  - 아키텍처 설계는 개발 중인 시스템에 대한 아키텍처를 결정하는 의사 결정 과정이다.
- **컴포넌트** (서브시스템, 모듈과 혼용되기도 하며, 교재에서는 모듈과 구분하여 정의함)  
컴포넌트란 명백한 역할을 가지고 있으며 독립적으로 존재할 수 있는 시스템의 부분이다.  
같은 기능을 가진 다른 컴포넌트로 대체 시킬 수 있다.
- **모듈**  
모듈이란 프로그래밍 언어의 문법 구조에서 정의된 컴포넌트를 말한다.
  - 예) 메소드, 클래스, 패키지는 Java 프로그램의 모듈이다.
  - 예) C 프로그래밍 언어에서의 모듈의 파일과 함수이다.



## ④ 아키텍처 설계의 비교 (Ontologies 소프트웨어 제품)

- 왼쪽은 3<sup>rd</sup> 파티 컴포넌트(DBMS)가 변경되면, 응용 컴포넌트들이 모두 변경되어야 함
- 오른쪽은 래퍼(wrapper)를 사용하여, DBMS가 변경되어도 래퍼 I/F만 변경하면 됨



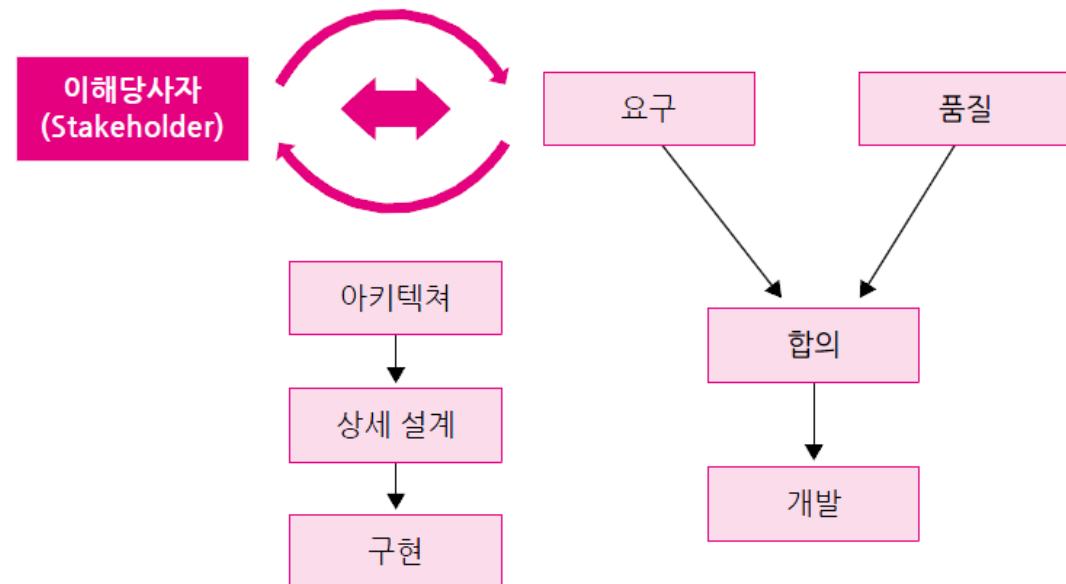
→ Ontologies는 복잡성/비효율성을 고려하여 초기에 왼쪽 구조를 선택하여, 개발팀에 내분이 일어났고, 결국 프로젝트가 실패함



## 비기능적 요구에 의한 제약 사항

- **기술적 제약**: 애플리케이션이 사용하는 특정 기술을 명시하여 선택을 제한  
(예: Java 개발자만 있으니, Java 언어로 개발해야 한다.)
- **비즈니스 제약**: 기술적 측면이 아닌 경영적 측면의 제약  
(예: 잠재적 고객 확보를 위해 XYZ 도구에 대한 인터페이스를 제공해야 한다.)
- **품질 제약**: 확장성, 가용성, 변경용이성, 이식성, 서용성, 성능 등에 대한 제약

## 아키텍처 설계와 품질의 도식화





## 품질 제약의 종류와 사례 (1/2)

품질	속성	설계 목표
성능	처리량 (throughput)	시스템이 단위 시간 당 얼마나 많은 작업을 하여야 하는가? 단위는 초당 처리한 트랜잭션 수(transaction per second) 또는 메시지 수(message 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 소모할 수 없다.
확장성 (scalability)	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 나타냄.
변경용이성	수정 비용	새로운 기능적 요구나 비기능적 요구를 충족하기 위하여 얼마나 쉽게 애플리케이션을 수정할 수 있는가?



## 품질 제약의 종류와 사례 (2/2)

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 가동 중인 비율. 대부분의 IT 애플리케이션은 항상 가동되어야 하므로 100%의 가동률을 요구한다.
통합	호환성	데이터의 호환성을 제공하거나 다른 시스템과의 인터페이스를 제공하는 것을 의미한다.





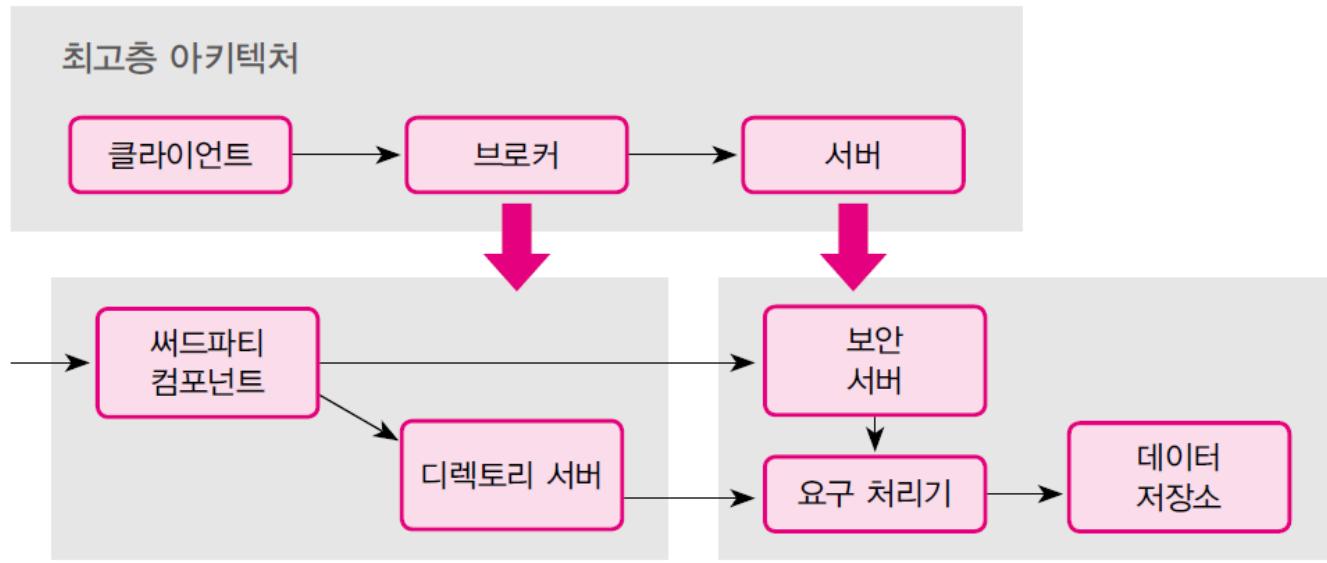
## 마키텍처(Marketecture)

- 아직 존재하기 않는 기술이나 상품을 홍보하기 위한 추상화된 구조
- 시스템의 구조나 동작을 한 페이지로 간략히 작성한 것으로, 이해당사자들이 시스템을 설계, 검토하는 데 유용하게 사용됨



## 아키텍처의 표현

- 컴포넌트를 블랙박스로 표현하고, 외부에 보일 수 있는 구조적 속성만 나타냄
- 아키텍처 표현의 가장 강력한 방법은 계층적 분할(hierarchical decomposition)임





## In this chapter ...

아키텍처 설계(Architecture Design)

6.1 아키텍처 설계란?

6.2 설계 원리

6.3 아키텍처 설계 과정

6.4 아키텍처 스타일

6.5 미들웨어 아키텍처

6.6 설계 문서화

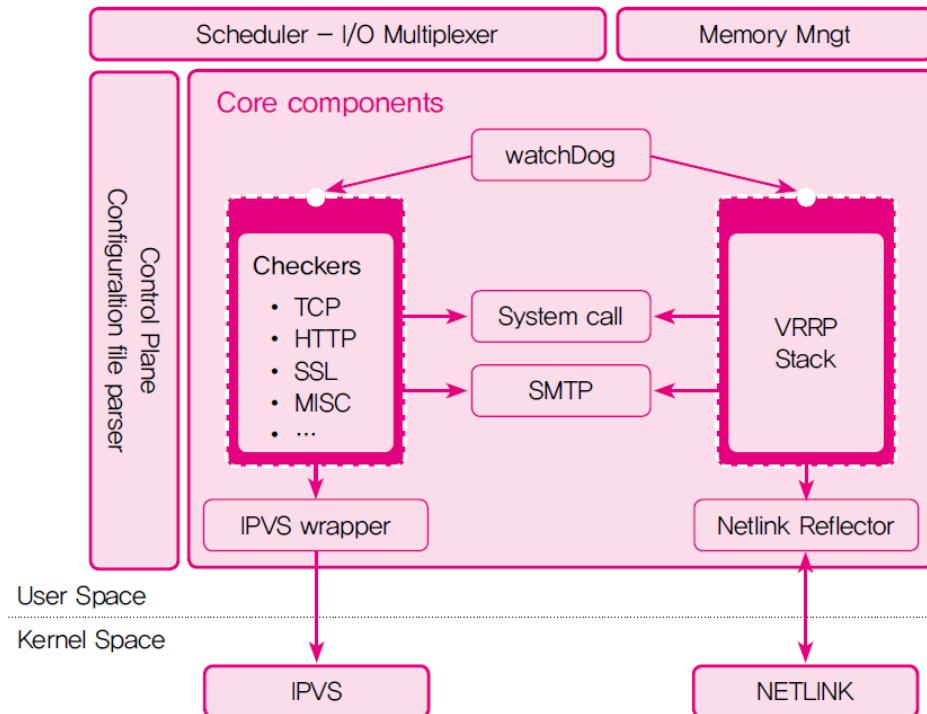


## ④ 시스템을 개발하는 조건이나 운용될 환경 조건의 제약 안에서 가능한 여러 설계 중에서 최적의 설계 안을 발견 하는 것

- 설계를 평가하기 위한 특성과 기준을 명시하여야 한다. (정량적)
- 효율성(Efficiency): 시스템이 사용하는 자원이 적정하고 효과적임을 의미한다.
- 단순성(Simplicity): 이해하기 쉬운 설계를 작성해야 한다.

## ⑤ 소프트웨어 설계의 중심이 되는 원리 (뒤에 설명)

- 추상화(Abstractation)
- 정보은닉(Information hiding)
- 단계적 분해(Stepwise refinement)
- 모듈화(Modularization)



단순성	효율성	분할, 계층화	추상화	모듈화
복잡한 여러 가지 요소를 교통 정리 하여 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.



## 주요 설계 원리

- **문제의 분할, 단계적 분해(stepwise refinement):**

처음엔 간단히, 차츰 세밀하게...

(→ 사칙연산 → 더하기 → 정수 더하기)

- **추상화(abstraction):**

복잡한 문제를 일반화하여, 쉽게 이해할 수 있도록 하는 원리

(→ 피타고라스 정리?)

- **모듈화(modularization):**

모듈은 독립성을 가져야 함...

(→ 많은 라이브러리 함수를 생각하세요, 삼각함수 등)

- **정보 은닉(information hiding):**

함수의 내부 변수 사용이 외부에 알려질 필요가 없다...

(→ 내정 간섭하지 마요~)

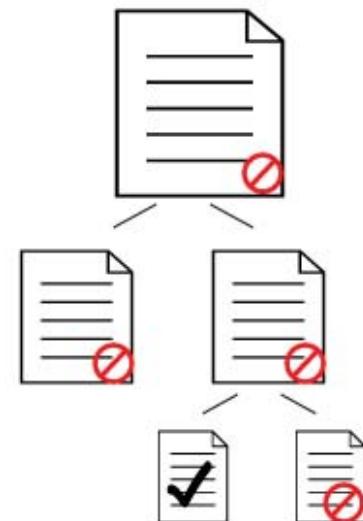


## ▣ 분할과 정복 (divide and conquer)

- 큰 문제를 한 덩어리로 다루기는 어려움
- 큰 문제를 작은 조각으로 나누고, 각 조각을 하나씩 해결(정복)함

## ▣ 분할의 결과

- 장점: 작은 조각(컴포넌트)은 해결이 용이함
  - 단점: 여러 조각에 따른 통신 오버헤드가 발생함
- ➔ 설계자는 분할을 중지할 시점(단위)을 결정해야 함





- 기능을 최대한으로 떼어내어 생각
- 점진적으로(incrementally) 구체화
- 상세한 내역(알고리즘, 자료구조)는 가능한 뒤로 미룸

추상화 I



추상화 II

CAD software tasks:  
use interaction task;  
**2-D drawing creation task;**  
graphics display task;  
drawing file management task;  
end.

추상화 III

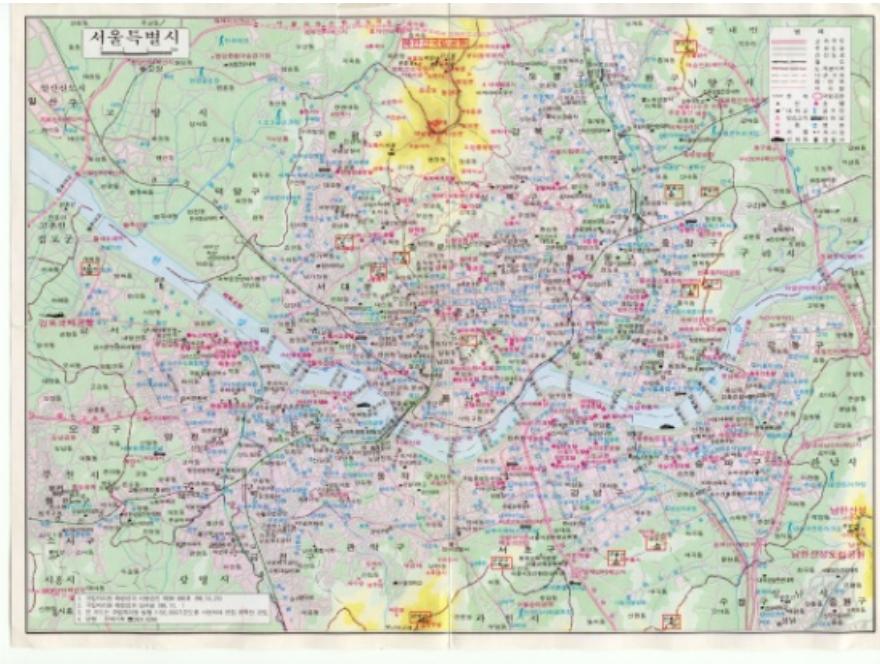
```
procedure: 2-D drawing creation;  
repeat until <drawing creation task terminates>  
  do while <digitizer interaction occurs>  
    digitizer interface task;  
    determine drawing request;  
    line: line drawing task;  
    circle: circle drawing task;  
    .  
    .  
    .
```



# 추상화 (1/2)

아키텍처 설계(Architecture Design)

▣ (현실의) 복잡한 문제 → 추상화 → 개념 정립



▣ 소프트웨어의 구조를 이루는 계층의 파악



## 기능 추상화

- 입력자료를 출력자료로 변환하는 과정을 추상화
- 부프로그램(subprogram)의 목적과 기능만 생각



## 자료 추상화

- 자료와 기능을 묶어서 생각 (data object 구성하는 방법) → OO Concept



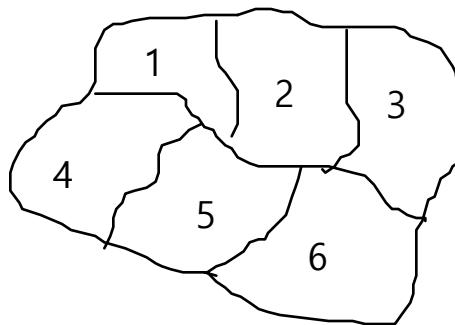
## 제어 추상화

- 외부 이벤트에 대한 반응을 추상화 → ... 입력되면 ...을 처리하고...

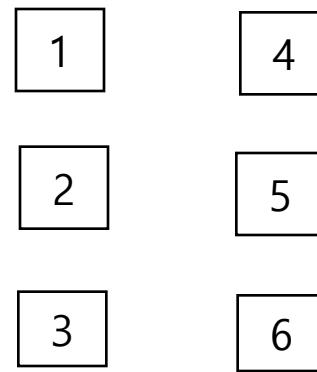


# 모듈화(Modularization) (1/6)

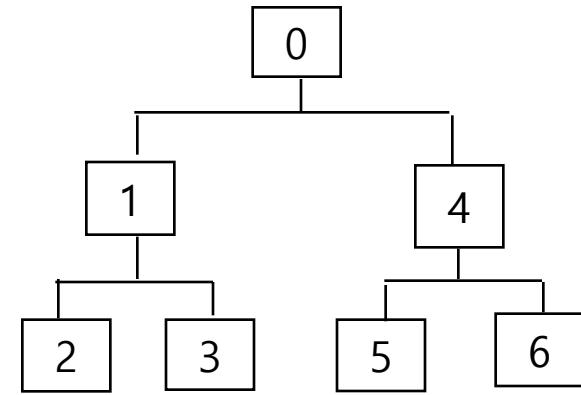
아키텍처 설계(Architecture Design)



문제영역



시스템 분해



시스템 구조

▣ 시스템 분해를 어떻게 해야 할 것인가? → 모듈로 분해한다.

▣ 한 모듈의 규모는 어떠해야 하는가?

- 30 lines ~ 50 lines ???
- 너무 길면 이해하기 어렵고, 너무 짧으면 성능이 저하됨



## 모듈의 이식성(portability)

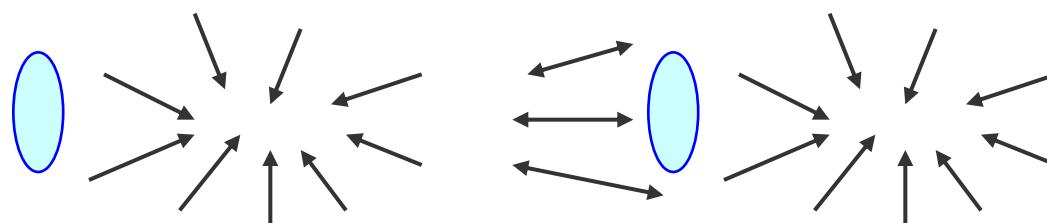
- 특정 환경에서만 동작하는 것이 아니라 일반적 환경에서 동작하면 이식성이 높다고 이야기한다.
- 이식성이 높은 S/W를 개발하는 것은 모든 S/W Engineer의 공통된 목표이다.



## 모듈을 어떻게 만들 것인가? 어떻게 모듈을 구성할 것인가?

- 모듈(내) 응집력(intra relationship)은 강하게,  
→ 모듈(간) 결합력(inter relationship)은 약하게

- 모듈의 응집력: 모듈 내 기능/요소들이 갖는 관계
- 모듈의 결합력: 모듈 간의 관계





## 모듈의 응집력(Cohesion)

- 하나의 모듈은 전체 시스템이 갖는 여러 기능 중에 하나의 기능을 갖도록 설계해야...
- 모듈 내의 모든 요소는 하나의 목적을 가지고 있는 것이 바람직하다.
- Theme sentence: 한 문단의 주제를 담고 있는 문장 (vs. support sentence)

## 모듈의 응집력 구분

### 기능적 응집(functional cohesion):

잘 정의된 하나의 기능이 하나의 모듈을 이룬 경우

예) "판매 세금 계산" → (동사, 목적어) 한 쌍으로 구성

### 순차적 응집(sequential cohesion):

모듈 내 한 작업의 출력이 다른 작업의 입력이 되는 경우

예) "다음 거래를 읽고, 그 결과를 마스터 파일에 반영함"

### 교환적 응집(communication cohesion):

동일한 입력과 출력을 사용하는 여러 작업들이 모인 경우

예) "인사 기록 파일에 근무 성적을 기재하고, 급여를 갱신함"



응집력이 강함



응집력이 약함



## 모듈의 응집력 구분 (계속)

- **절차적 응집(procedural cohesion):**

공유하는 것은 없으나, 큰 테두리 안에서 같은 작업에 속하는 경우

예) "총계를 출력하고, 화면을 지우고, 메뉴를 뿌리고, 메뉴 선택 코드를 받음"

응집력이 강함

- **시간적 응집(temporal cohesion):**

특정 시간에만 수행되는 기능을 묶어놓은 모듈

예) 초기화 루틴(변수 할당, 초기값 설정, ...)

- **논리적 응집(logical cohesion):**

유사 성격을 갖거나 특정 형태로 분류되는 처리 요소들을 하나의 모듈로 형성

예) "사칙연산에서 주어진 매개변수에 따라 다른 계산을 함" → 연산간 관계가 없음

- **우연적 응집(coincidental cohesion):**

아무 관련 없는 처리 요소들로 모듈이 형성된 경우

→ 모듈 개념이 상실되어 이해 및 유지보수가 힘든 단점이 있음

응집력이 약함



## 모듈의 결합도(Coupling)

- 모듈간의 상호 의존하는 정도를 의미한다.
- 모듈은 하나의 블랙 박스로 다른 모듈과의 독립성이 높아야 한다.
- 독립적인 모듈이 되기 위해서는 다른 모듈과의 결합도가 약해야 한다(loosely coupled).

## 모듈의 결합도 구분

### 자료 결합(data coupling):

모듈 간의 인터페이스가 자료 요소(파라메터)로만 구성된 경우

예) add(3, 5), sort(a)



결합도가 약함

### 스탬프 결합(stamp coupling):

모듈 간의 인터페이스를 통해 배열, 레코드가 전달되는 경우

(단, 배열, 레코드 내용 전체가 사용되면 “자료 결합”으로 볼 수 있음)

예) print\_salary(인사 기록 레코드)

결합도가 강함





## 모듈의 결합도 구분 (계속)

- **제어 결합(control coupling):**

한 모듈이 다른 모듈에게 제어 요소(function code, switch, flag 등)를 전달하는 경우  
예) integer\_operation('+', 3, 5)

- **공통 결합(common coupling):**

공통된 자료 영역을 사용하는 경우  
→ 자료 영역의 보호가 어렵고, 자료 구조 변경 시 파급 효과가 큼  
예) C/C++ 등에서 global 변수를 사용하는 예제, Shared Memory 사용

결합도가 약함

- **내용 결합(content coupling):**

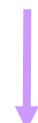
한 모듈이 다른 모듈의 일부분을 직접 참조 또는 수정하는 경우

결합도가 강함

→ 공유 부분을 변경할 필요가 생기면 그 파급 효과가 가장 큼

예 1) 에셈블리어에서 한 모듈이 다른 모듈의 데이터를 참조하는 경우

예 2) 한 모듈에서 다른 모듈로 분기(goto)하는 경우





# 정보 은닉(Information Hiding)

아키텍처 설계(Architecture Design)

- 각 모듈의 자세한 처리 내용이 시스템의 다른 부분으로부터 감추어져 있어야 ( $\rightarrow$  내부적으로 어떤 변수를 쓰던...)



- 각 모듈이 다른 모듈에 구애 받지 않고 설계  
 $\rightarrow$  I/F만 잘 정의하자.

- 인터페이스가 모듈 안의 구체적 사항을 최소로 반영
  - 전역변수가 없어야
  - 모듈의 입력이 1이면 입력, 2이면 출력, ...  $\rightarrow$  좋은 모듈 설계가 아님  $\rightarrow$  모듈화와 연관

- 모듈 단위의 수정, 시험, 유지보수에 큰 장점
  - 모듈 설계 평가에 기초
  - 모듈을 독립적으로 시험할 수 있으며, 모듈 별로 개선 및 최적화 할 수 있음



## In this chapter ...

아키텍처 설계(Architecture Design)

6.1 아키텍처 설계란?

6.2 설계 원리

6.3 아키텍처 설계 과정

6.4 아키텍처 스타일

6.5 미들웨어 아키텍처

6.6 설계 문서화



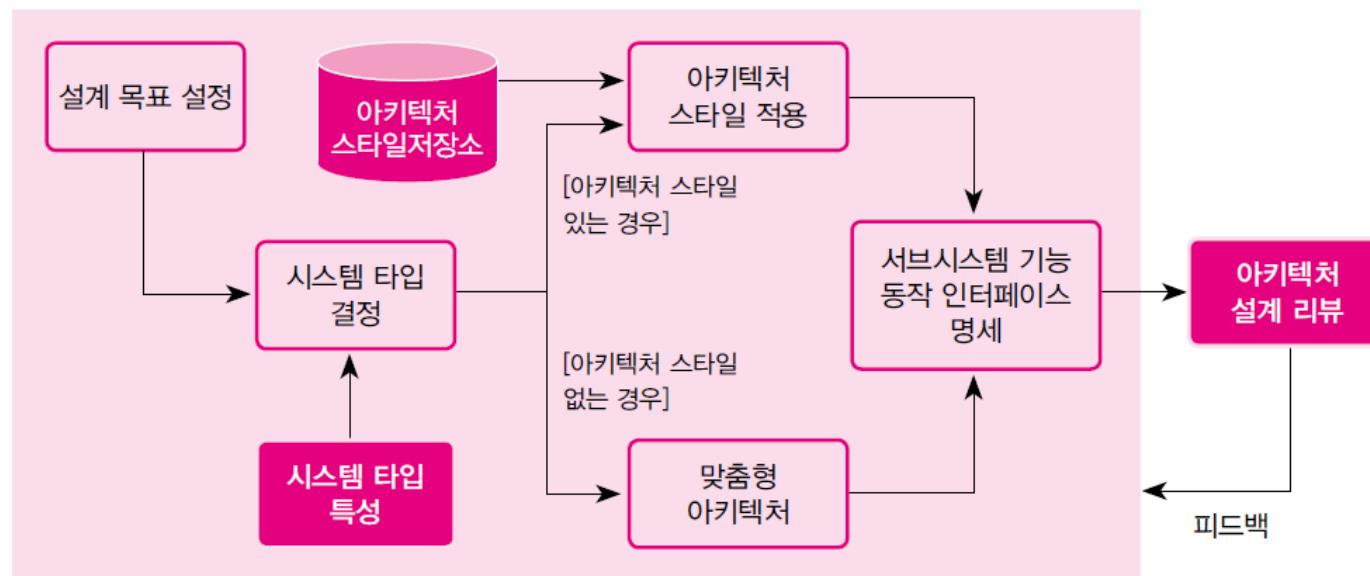
**Step 1** 설계 목표를 설정한다.

**Step 2** 시스템의 타입을 결정한다. (→ 아키텍처 스타일을 선택한다.)

**Step 3** 아키텍처 스타일을 적용하거나 아키텍처 설계를 커스터마이즈 한다.

**Step 4** 서브시스템의 기능, 인터페이스 인터랙션 동작을 작성한다.

**Step 5** 아키텍처 설계를 검토한다.





④ 아키텍처 설계는 시스템의 전반적인 구조를 결정하는데 목적이 있다.

④ 아키텍처 설계 목표는

설계에 의하여 달성되어야 할 시스템의 속성 또는 관점을 명시하며,  
품질과 보안 요구가 포함된 소프트웨어 요구로부터 파생된다.

④ 아키텍처 설계에서 고려해야 할 요구 사항

- 변경, 유지보수 용이성
- 상용 컴포넌트의 사용
- 시스템 성능
- 신뢰성
- 보안
- 고장 인내성
- 복구



## 대화형(interactive) 시스템

- PC와 웹 애플리케이션이 널리 사용되면서 가장 보편적임 됨
- 액터가 구동 시키는 비즈니스 프로세스를 나타내는 사용 사례로부터 모델링

## 이벤트 중심(event-driven) 시스템

- 상태에 의존적이며 반응 동작을 보임 (예: PTT(Push To Talk)에서 반응 시간 등)
- 임베디드 시스템에서 주로 쓰임 – 불규칙적인 이벤트를 적절히 제어하기 위함

## 변환(transformational) 시스템

- 입력을 출력으로 변환하는 정보처리 작업으로 구성됨
- 과학 계산, 엔지니어링 계산, 데이터 분석 등의 시스템에서 사용됨

## 객체 영속(object-persistence) 시스템

- 데이터베이스나 파일 시스템에 객체를 저장하고 검색할 수 있는 능력을 가진 시스템
- DBMS를 사용하는 정보 저장/처리 시스템에 해당



## ④ 아키텍처 표현 방법은 관점에 따라 다양함

### ① 모듈

- 시스템을 단위 코드의 집합으로 보는 관점 (
- 단위 코드는 시스템 기능 중 일부를 구현한 것임

### ② 컴포넌트와 커넥션

- 시스템을 컴포넌트라 부르는 런타임 개체의 집합으로 보는 관점
- 컴포넌트는 실행 시스템에서 식별 가능한 고유 단위

### ③ 배치

- 단위 소프트웨어가 어떤 하드웨어 노드에 배치되는가에 관점



## 아키텍처의 세 가지 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	



## In this chapter ...

아키텍처 설계(Architecture Design)

6.1 아키텍처 설계란?

6.2 설계 원리

6.3 아키텍처 설계 과정

6.4 아키텍처 스타일

6.5 미들웨어 아키텍처

6.6 설계 문서화



## 건축 설계로 본다면...

- 설계와 시공에 대한 가이드가 될 큰 밀그림
- 일관적인 모양과 조화를 위한 스타일을 정하는 작업



## 스타일이라는 개념을 소프트웨어 구조에도 적용

- 어떻게 동작해야 할 지의 동작 메커니즘의 큰 그림을 결정
- 중앙 DB를 두고? C/S 모델로?



## 일단 시스템이 개발된 뒤에는 잘못된 구조를 바로잡기가 쉽지 않음

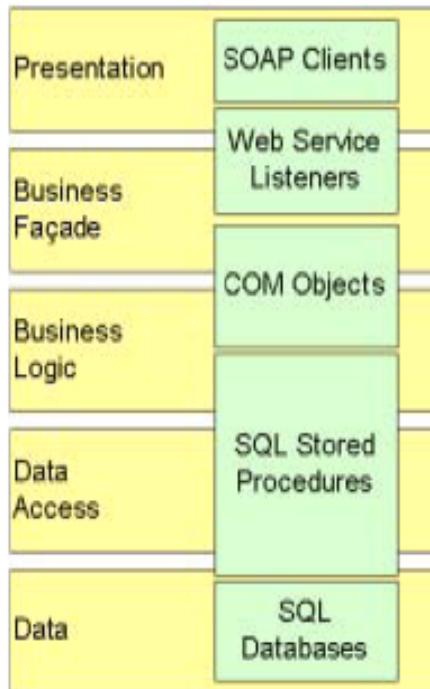


## 소프트웨어 구조는 시스템 분할, 전체 제어 흐름, 오류 처리 방침, 서브시스템 간의 통신 프로토콜을 포함

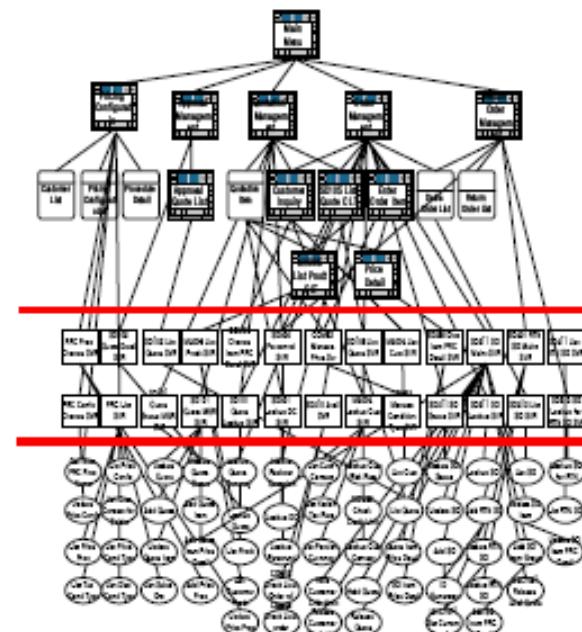


## 좋은 구조와 나쁜 구조의 차이

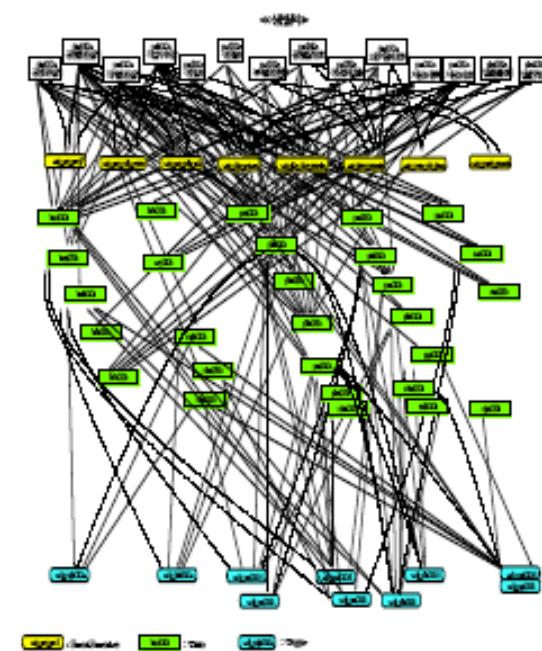
- 소프트웨어 개발 과정은 물론 최종 제품의 품질에 광범위한 영향을 미침



.Net Web Service  
Architecture



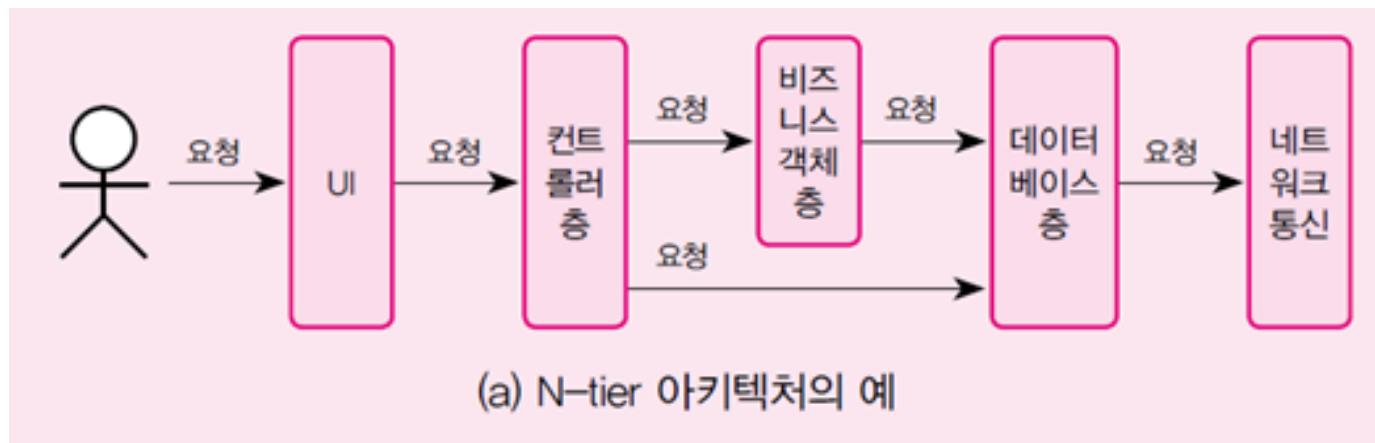
Project A+  
(MVC Layered Architecture)



Project A  
(No Architecture)



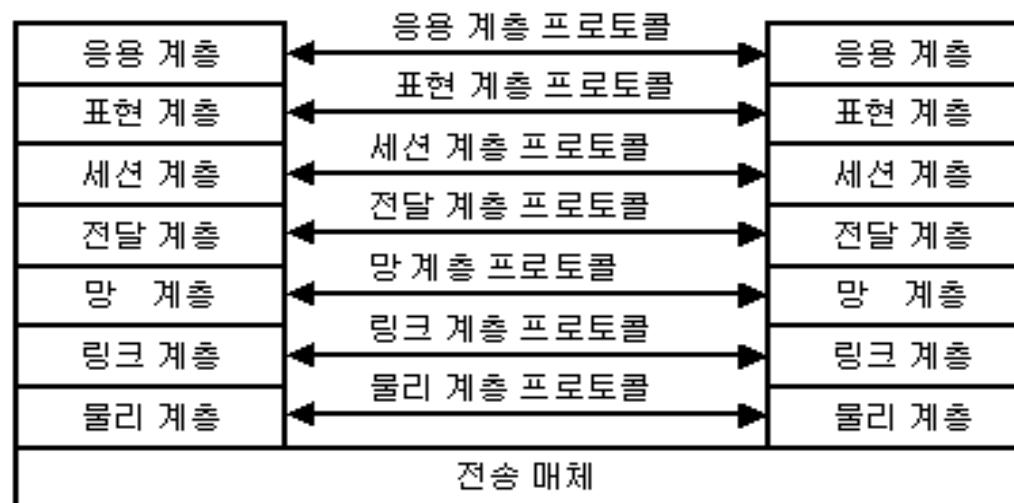
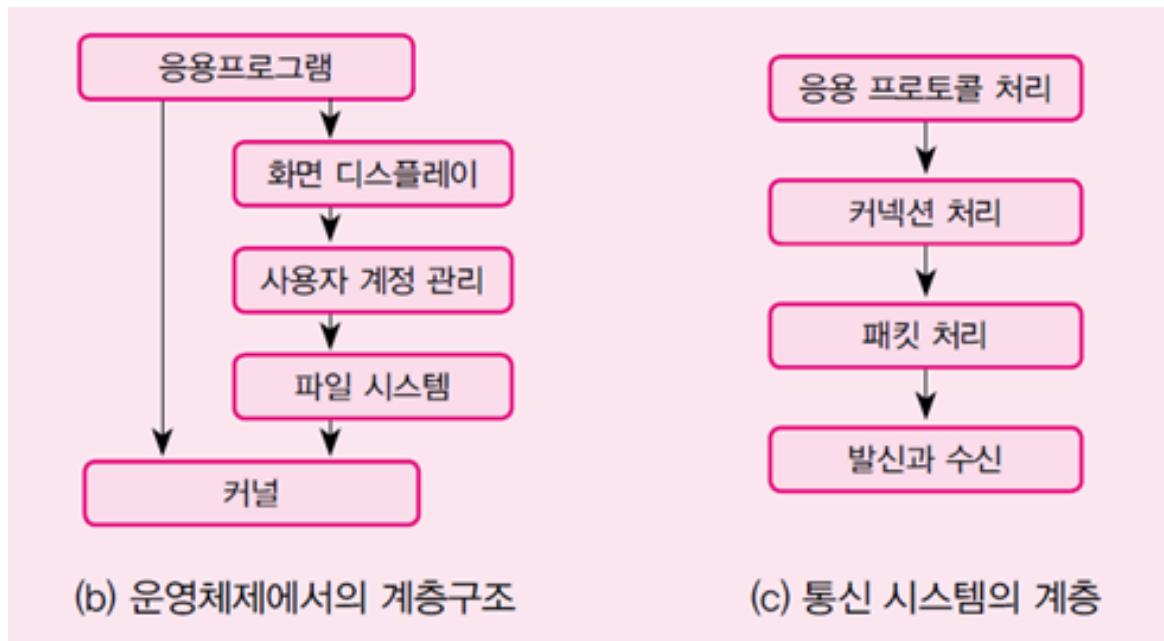
- ▣ 각 서브 시스템이 하나의 계층이 되어 하위 층이 제공하는 서비스를 상위 층의 서브시스템이 사용
- ▣ 추상화의 성질을 잘 이용한 구조
- ▣ 대표적인 예: 네트워크의 OSI 7 Layer 구조
- ▣ 장점: 각 층을 필요에 따라 쉽게 변경할 수 있음
- ▣ 단점: 성능 저하를 가져올 수 있음





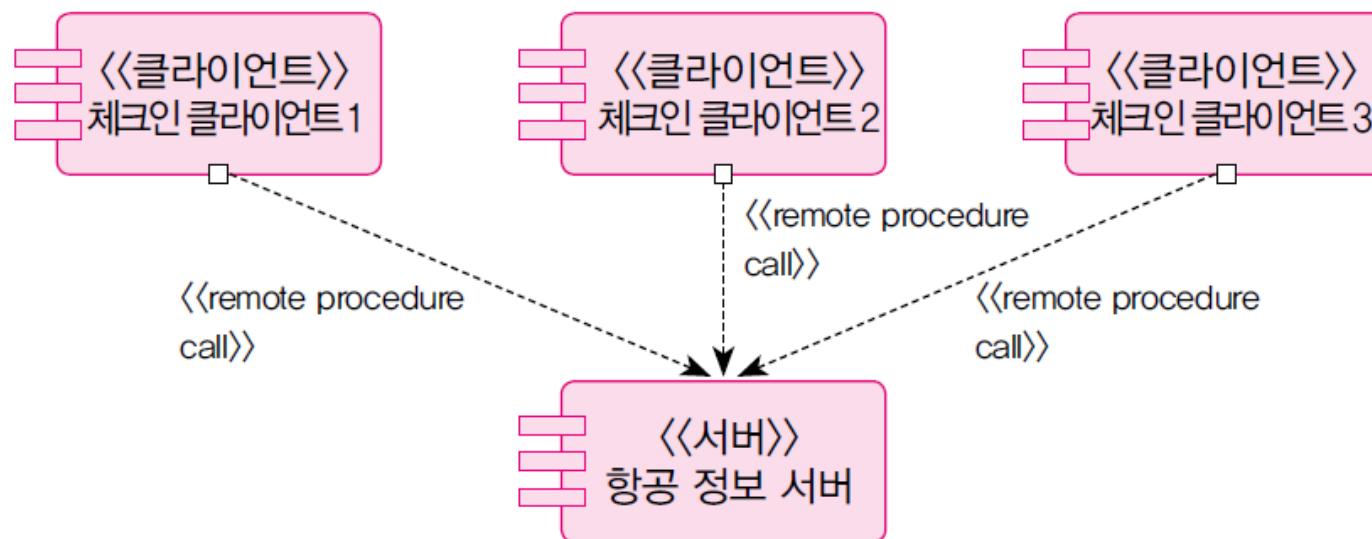
## 계층 구조 스타일 (2/2)

아키텍처 설계(Architecture Design)





- 서버와 여러 개의 클라이언트로 구성
- 요청과 결과를 받기 위하여 동기화 되는 일을 제외하고는 모두 독립적임
- 특정 서브시스템이 다른 서브시스템에 서비스를 제공하도록 지정할 때 유용함
- 대부분의 웹 기반 애플리케이션과 파일 서버, 전송 프로토콜 포함





## 트랜잭션 처리 아키텍처는 입력을 하나씩 읽어 처리한다.

- 입력은 시스템에 저장되어 있는 데이터를 조작하는 명령들, 즉 트랜잭션이다.
- 트랜잭션 처리 시스템은 서버에 탑재되며, 데이터베이스 엔진에 대표적 사례이다.

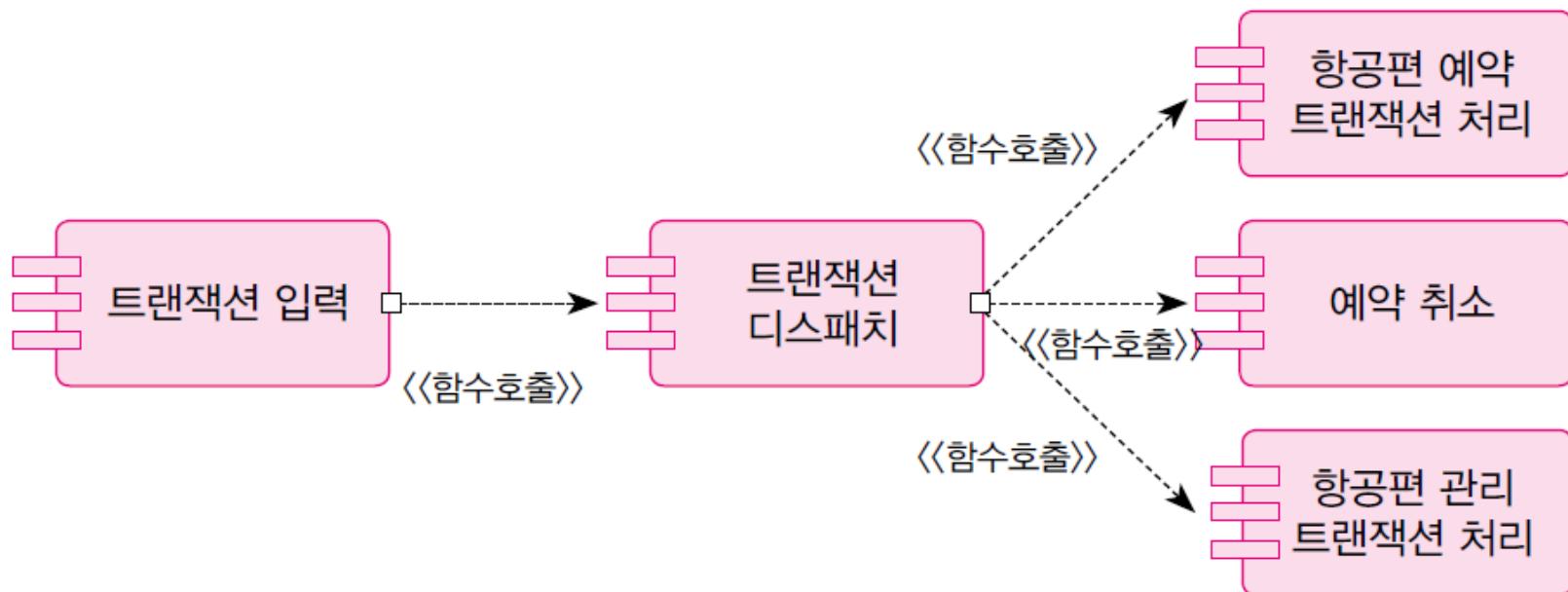
## 트랜잭션을 어디서 처리하는지 결정하는 **디스패처(dispatcher)**라고 하는 교통정리 컴포넌트가 필요하다.

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

## 대부분의 트랜잭션 처리 시스템은 여러 스레드나 프로세스가 트랜잭션을 동시에 처리하는 환경에서 작동한다.



## 항공권 예약 시스템의 사례





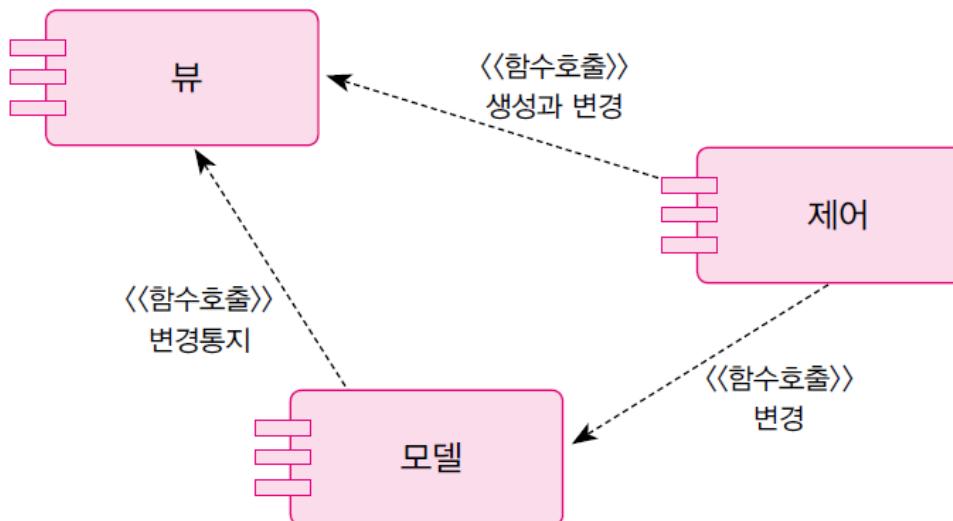
## MVC(Model, View, Control)

- 모델 서브시스템: 도메인의 지식을 저장 및 보관
- 뷰 서브시스템: 사용자에게 보여줌
- 제어 서브시스템: 사용자와의 상호 작용을 관리



## 분리하는 이유

- 사용자 인터페이스, 즉 뷰와 제어가 도메인 지식을 나타내는 모델보다는 더 자주 변경될 수 있기 때문임

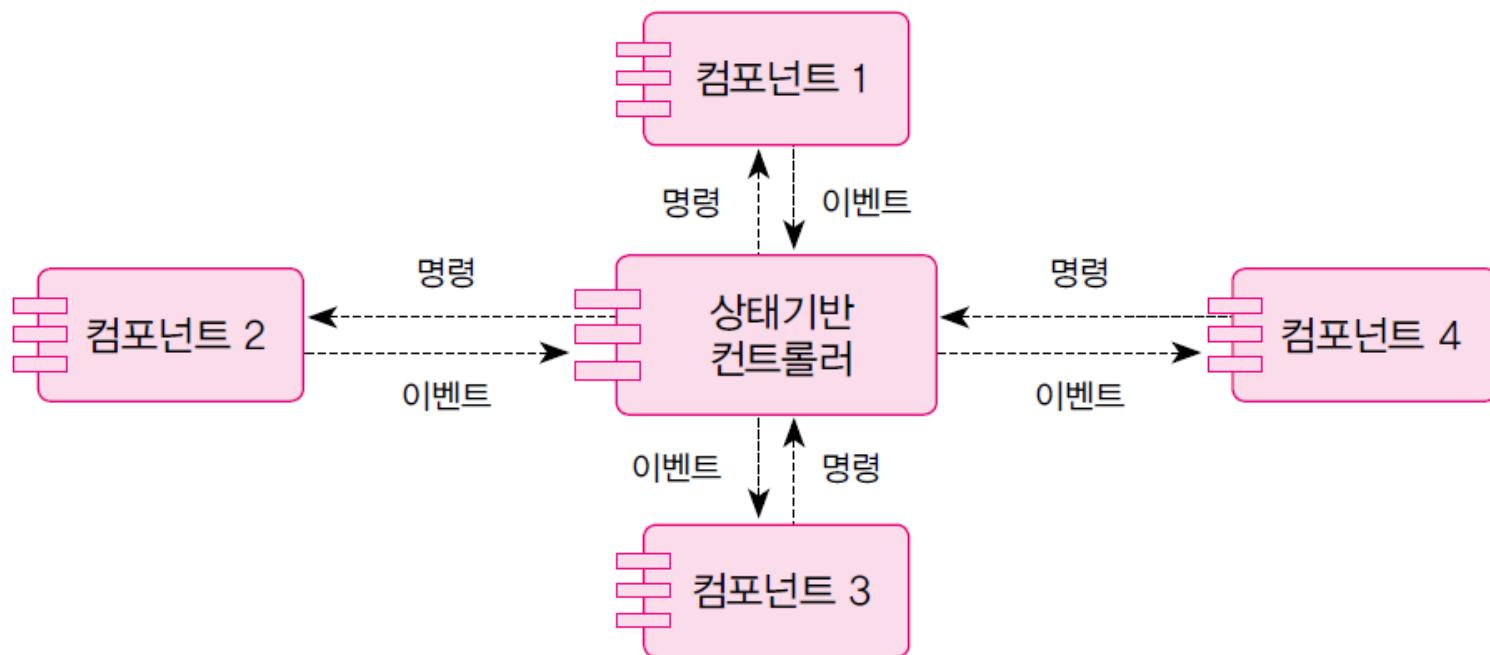


### Excel에서

- 사용자는 표, 그래프, 차트 등으로 볼 수 있으나(뷰),
- 실제 저장은 한 군데 있으며(모델),
- 다른 뷰에서 각기 수정하여도(제어) 한 군데 반영됨



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





- 비즈니스 시스템은 데이터베이스에 객체를 저장하고 나중에 이를 검색 할 필요가 있음

- 관점의 분리, 정보 은닉, 변경이 쉬운 설계, 단순화 등의 특징을 제공
- 다른 종류의 DB, 다른 제품의 DB가 사용되어도 사용 및 유지보수가 단순화됨





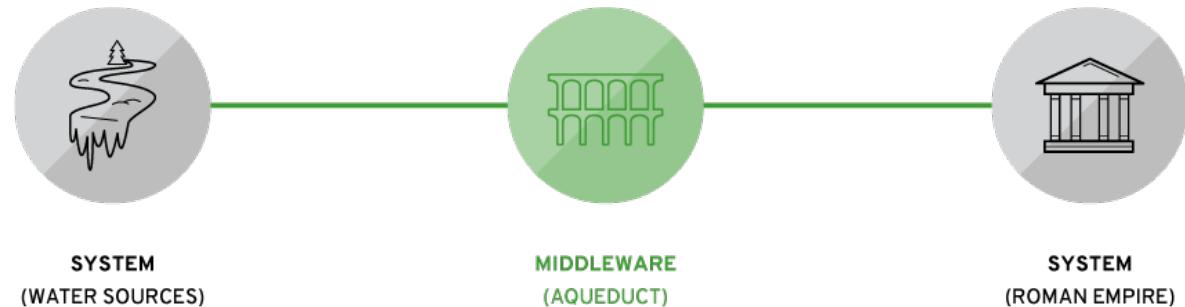
## In this chapter ...

아키텍처 설계(Architecture Design)

- 6.1 아키텍처 설계란?
- 6.2 설계 원리
- 6.3 아키텍처 설계 과정
- 6.4 아키텍처 스타일
- 6.5 미들웨어 아키텍처**
- 6.6 설계 문서화



- 소프트웨어 컴포넌트들을 연결하기 위해 준비된 인프라 구조 제공
- 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 (미들웨어 존재를 인지할 필요 없이) 애플리케이션과 상호작용한다.
  - (통상 미들웨어 역할을 알 필요가 없으나) 이를 알아야 하는 유일한 경우는 고장이다.





비즈니스 프로세스 관리자

BizTalk, TIBCO StaffWare, ActiveBPEL

메시지 브로커

Mule, WebSpere Message Broker

애플리케이션 서버

JEE, CCM, .NET

트랜스포트

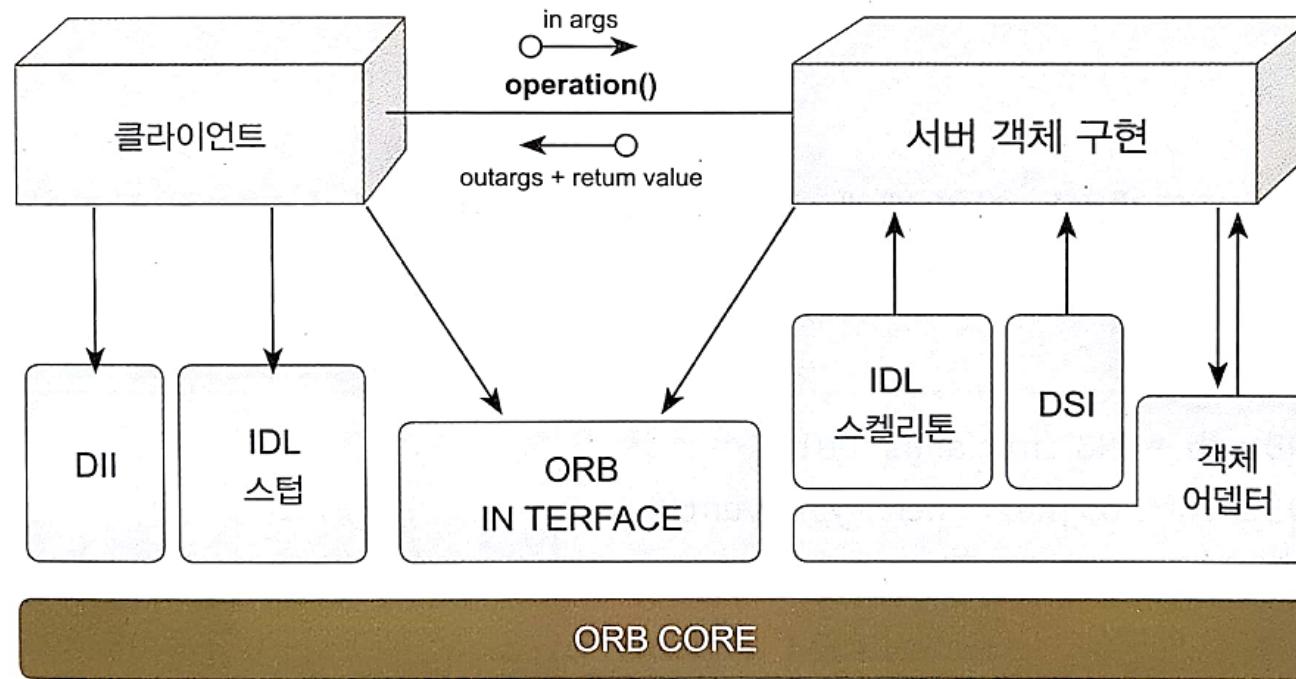
메시지 기반 미들웨어, 분산객체 시스템,  
SOAP

- ▣ **트랜스포트**: 소프트웨어 컴포넌트 사이에 요구를 보내고 데이터를 이동하기 위한 기본 패이프 역할을 한다.
- ▣ **애플리케이션 서버**: 트랜잭션, 보안, 디렉토리 서비스 등의 응용 서비스를 제공한다.
- ▣ **메시지 브로커**: 빠른 메시지 전달과 애플리케이션 컴포넌트 사이에 메시지 교환, 라우팅 기능을 제공한다.
- ▣ **비즈니스 프로세스 관리자**: 메시지 브로커 기능에 워크플로 스타일 애플리케이션 기능을 추가한 것이다.



분산 객체 기술은 미들웨어 기술 중의 하나로, CORBA로 알려진 분산 객체 기술이 1990년대부터 널리 사용되었다.

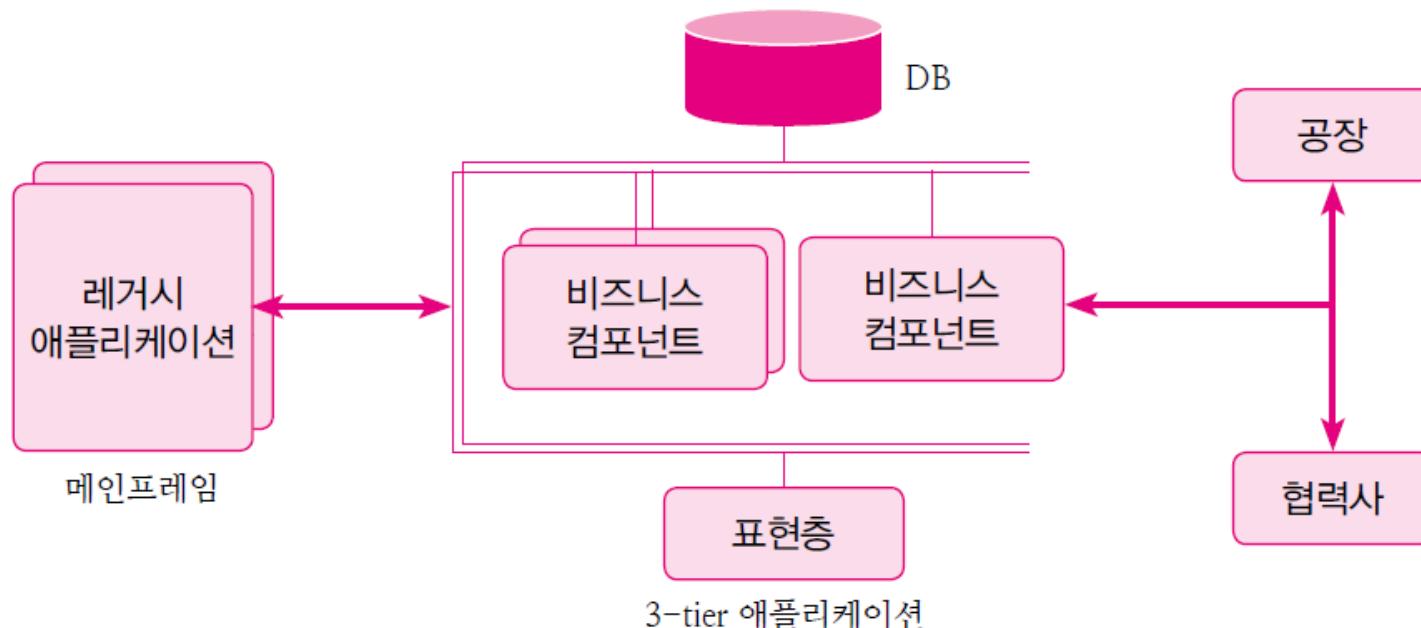
- **CORBA**: Common Object Request Broker Architecture
- 클라이언트는 ORB(Object Request Broker)를 통해 서버에 요청을 보낸다.
- 서버는 CORBA IDL(Interface Definition Language)를 통해 객체 인터페이스를 지원한다.





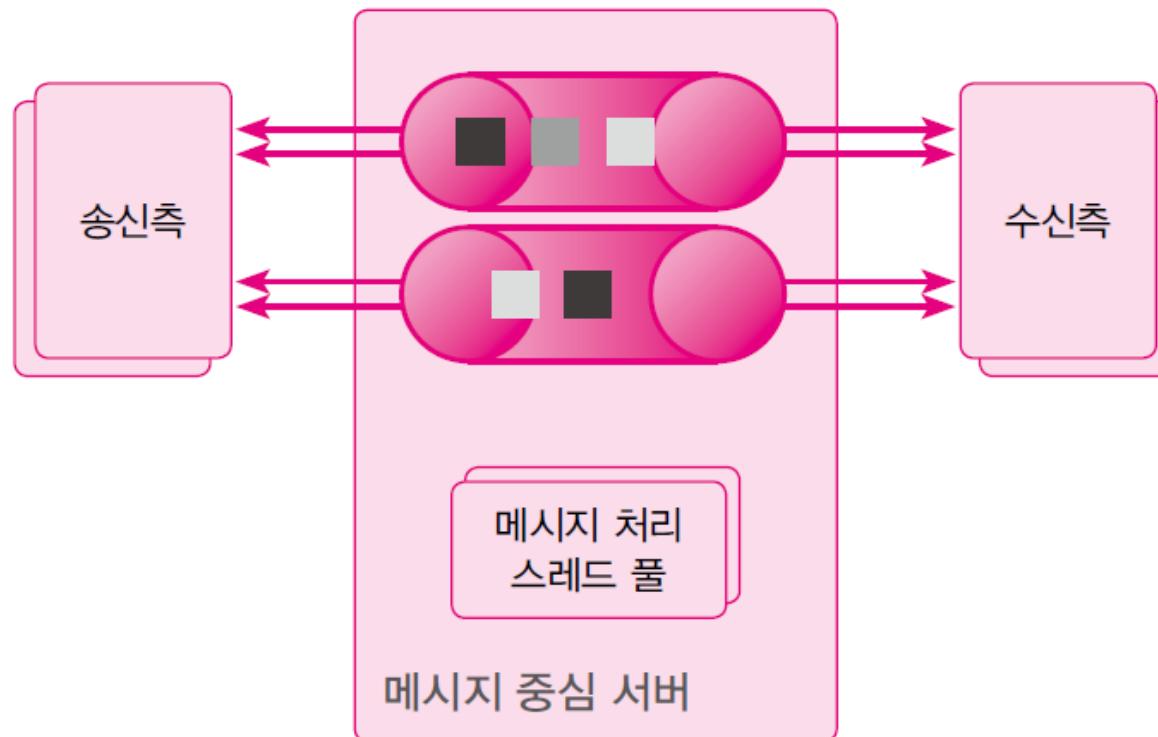
대규모 엔터프라이즈 시스템을 구축할 때 중요한 기술이다.

- 서로 다른 독립적인 애플리케이션을 하나의 통합된 시스템으로 묶기 위한 접착제 같은 역할을 한다.
- CORBA와 같이 메시지 송신측과 수신측이 강하게 결합되어 있지 않으며, 여러 애플리케이션을 다양한 기술과 다양한 플랫폼을 사용하여 구축할 수 있다.





## 메시지 중심 서버(Message-Oriented Server)





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

■ 웹 애플리케이션을 위한  
N-tier 아키텍처 사례  
(비즈니스 컴포넌트 층이 해당)

브라우저

웹서버

EJB.NET 컴포넌트

데이터베이스 ERP





## In this chapter ...

아키텍처 설계(Architecture Design)

- 6.1 아키텍처 설계란?
- 6.2 설계 원리
- 6.3 아키텍처 설계 과정
- 6.4 아키텍처 스타일
- 6.5 미들웨어 아키텍처
- 6.6 설계 문서화



## 아키텍처 문서를 작성하는 이유

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



## 설계문서는 다음 세 그룹이 의사 교환하는데 사용됨

- 설계를 구현할 사람 즉 프로그래머
- 미래에 설계를 변경할 엔지니어
- 설계된 시스템과 인터페이스 할 다른 시스템 또는 서브시스템을 개발하는 엔지니어



## 설계 문서가 일반적으로 포함하는 정보

(+ 실제 프로젝트 환경에 맞는 보다 자세하고 일관된 포맷을 사용하여야 함)

- (1) 목적
- (2) 우선순위
- (3) 설계의 아웃라인
- (4) 설계 이슈
- (5) 설계 상세 사항

## 설계문서에서 제외할 사항

- 훈련된 프로그래머나 설계자에게 당연히 여겨지는 정보를 문서화 하는 것은 피한다.
- 코드의 코멘트 안에 포함되어 있는 것이 더 좋을 내용은 설계 문서에서 피한다.
- 코드에서 자동으로 추출될 내용, 예를 들면 public 메소드의 리스트는 설계문서에서 제외한다.



# In this chapter ...

아키텍처 설계(Architecture Design)

- 6.1 아키텍처 설계란?
- 6.2 설계 원리
- 6.3 아키텍처 설계 과정
- 6.4 아키텍처 스타일
- 6.5 미들웨어 아키텍처
- 6.6 설계 문서화