5주 2강

소프트웨어 아키텍처

숭실사이버대학교의 강의콘텐츠는
저작권법에 의하여 보호를 받는바, 무단
전재, 배포, 전송, 대여 등을 금합니다.
*사용서체 : 나눔글꼴

이번 주차에는…

상위 설계 : 소프트웨어 아키텍처

- 소프트웨어 아키텍처

1. 소프트웨어 아키텍처

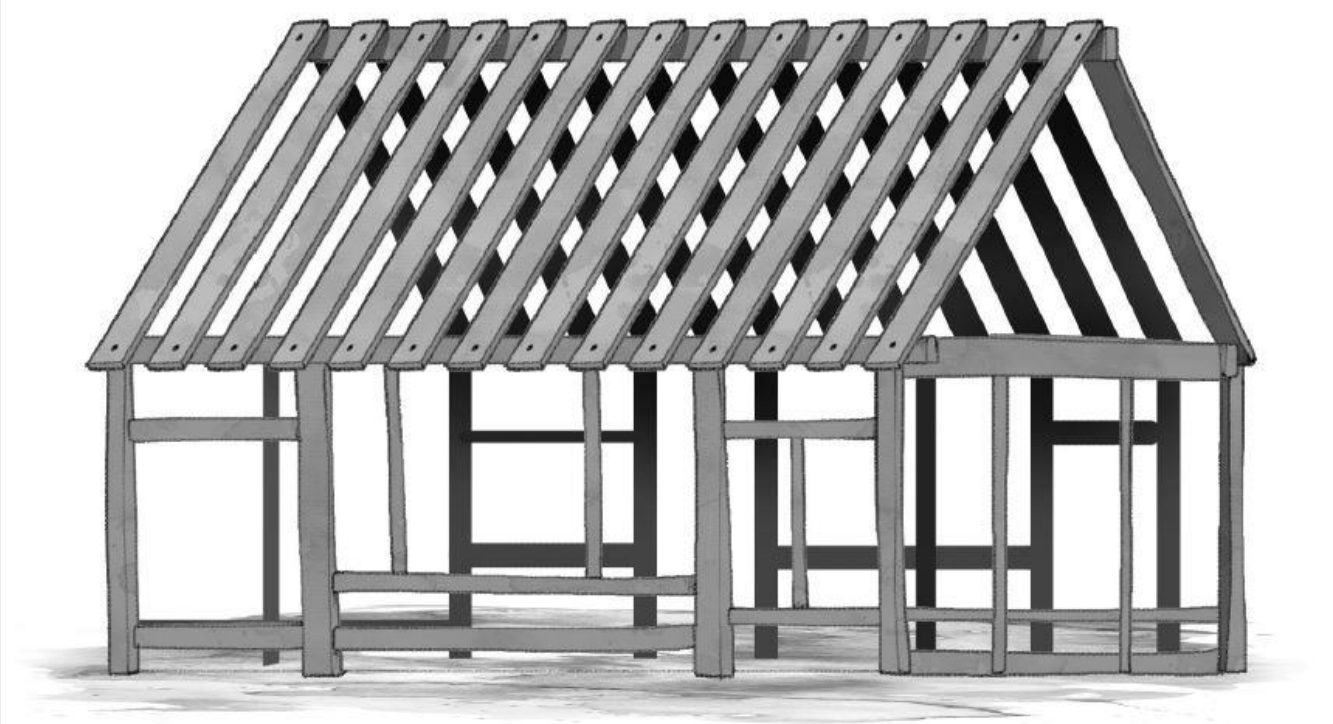

그림 5-14 건물 지을 때의 아키텍처

2. 아키텍처의 특징과 기능(1)

- 아키텍처의 정의
  - 구성 요소
  - 구성 요소들 사이의 관계
  - 구성 요소들이 외부에 드러내는 속성
  - 구성 요소들과 주변 환경 사이의 관계
  - 구성 요소들이 제공하는 인터페이스
  - 구성 요소들의 협력 및 조립 방법
- 소프트웨어 아키텍처
  - 개발할 소프트웨어 대한 전체적인 구조를 다룬다.
  - 소프트웨어를 이루고 있는 여러 구성 요소(서브시스템, 컴포넌트)를 다룬다.
  - 구성 요소들이 인터페이스를 통해서 어떻게 상호작용하는지를 정의해야 한다.
  - 세부 내용보다는 중요한 부분만을 다룬다.
  - 시스템 설계와 개발 시 적용되는 원칙과 지침이 있어야 한다.

3. 아키텍처의 특징과 기능(2)

- 아키텍처의 설계 시 고려 사항
  - 의사소통 도구로 활용할 수 있어야 한다.
  - 구현에 대한 제약 사항을 정의해야 한다.
  - 품질 속성을 결정해야 한다.
  - 재사용할 수 있게 설계해야 한다.
- 아키텍처의 설계 시 기술 방법
  - 이해하기 쉽게 작성
  - 명확하게 기술
  - 표준화된 형식 사용
  - 문서 버전 명시

4. 아키텍처의 품질 속성

- 품질 요구 사항
  - 시스템이 제공해야 하는 품질 속성의 수준
  - 가능하면 정확한 수치로 제시

- 소프트웨어 아키텍처
  - 이해 관계자들의 품질 요구 사항을 반영하여 품질 속성을 결정

5. 시스템의 품질 속성(1)

- 가용성(availability)
  - 시스템이 운용될 수 있는 확률로, 시스템이 장애 발생 없이 서비스를 제공할 수 있는 능력
  - 가용성을 높이려면: 하드웨어 이중화처럼 여분의 구성 요소를 포함하도록 설계
- 변경 용이성(modifiability)
  - 변경 요구 사항을 받았을 때 쉽게 변경할 수 있는 능력
  - 빈번하게 변경할 가능성이 높은 소프트웨어는 변경 용이성을 고려하여 아키텍처를 결정
- 성능(performance)
  - 사용자 요청과 같은 이벤트가 발생했을 때, 빠르고 적절하게 반응할 수 있는 능력
  - 공유 자원을 어떻게 사용하는지, 어떤 알고리즘을 사용해 구현하는지 등의 요소와 밀접

6. 시스템의 품질 속성(2)

- 보안성(security)
  - 허용되지 않은 접근에 대응할 수 있는 능력
- 사용성(usability)
  - 소프트웨어를 사용할 때 혼란스러워하거나 사용하는 순간에 고민하지 않게 하는 편의성
- 테스트 용이성(testability)
  - 사용자가 요구하는 기능을 만족스럽게 잘 수행하고 있는지를 얼마나 쉽고 철저하게 테스트할 수 있는지를 나타낸다.

7. 비즈니스 품질 속성(1)

- 시장 적시성time to market
  - 정해진 날짜에 소프트웨어를 출시해 경쟁력을 높일 수 있는 정도
- 비용과 이익cost and benefit
  - 비용을 더 들여 사용하고 효과를 볼 것인지, 아니면 비용을 절약하는 데 중심을 둘 것인지를 말한다.
  - 아키텍처를 설계 시: 비용을 더 많이 들여 유연한 설계를 할 것인지, 비용을 절감하는데 초점을 맞출 것인지 판단해야 함
- 예상 시스템 수명predicted lifetime of the system
  - 수명이 중요한 경우라면 변경 용이성, 확장성, 이식성을 더 중요하게 고려

8. 비즈니스 품질 속성(2)

- 목표 시장targeted market
  - 패키지 소프트웨어: 기능성 및 다양한 플랫폼에서도 잘 작동되어야 하므로 이식성을 충분히 고려한 설계 필요
- 신규 발매 일정 또는 공개 일정rollout schedule
  - 현재 버전에서는 기본 기능만 제공하고, 추후에 배포할 차기 버전에서 기능을 추가하여 완성도를 높일 예정이라면 유연성flexibility과 확장성을 고려한 설계 필요
- 기존 시스템과의 통합integration with legacy system
  - 아키텍처 설계 시 기존 시스템과의 통합 방법을 충분히 고려한 설계 필요

9. 아키텍처 품질 속성

- 개념적 무결성conceptual integrity
  - 개념적 무결성은 일관성이라고도 함
  - 전체 시스템과 시스템 구성 요소가 일관되도록 아키텍처를 결정

- 정확성과 완전성correctness and completeness
  - 사용자가 요구하는 기능을 충족시키는 정도로, 요구 분석 명세서와 일치하는 정도

- 개발 용이성(구축 가능성, buildability)
  - 전체 시스템을 적절한 모듈로 분할한 후 개발 팀에 알맞게 분배하여 개발함으로써 정해진 기간 내에 완성하고, 개발 과정 중에도 쉽게 변경할 수 있는 능력

10. 이해 관계자별 품질 속성

- 발주자 관점
  - 제품 가격(또는 개발비)이 중요, 응찰 시 가장 적게 써낸 업체 선정 확률 높음
- 사용자 관점
  - 완벽한 기능뿐만 아니라 사용하기 쉽고 빨리 이해할 수 있는 아키텍처의 속성을 요구
- 개발자 관점
  - 플랫폼이 달라져도 새로운 플랫폼에 쉽게 적용할 수 있는 아키텍처의 속성에 관심
  - 변경 요청 시 쉽게 변경할 수 있는 설계

11. 아키텍처 구축 절차(1)

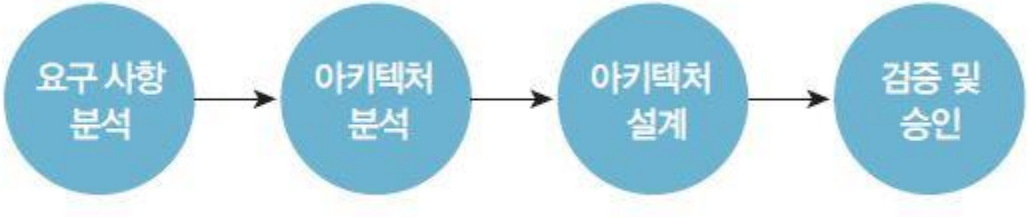

그림 5-18 아키텍처 구축 절차

- 요구 사항 분석
  - 소프트웨어 개발의 요구 사항 분석 단계와 같다.
  - 품질 속성과 같은 비기능적인 요구 사항에 더 많은 관심을 둠
    - 요구 사항 취득, 식별, 명세, 분류, 검증
    - 기능적/비기능적 요구 사항 분류 및 명세

12. 아키텍처 구축 절차(2)

- 아키텍처 분석

- 아키텍처 설계
  - 관점 정의: 이해 관계자 파악, 이해 관계자별 관점 정의
  - 아키텍처 스타일 선택: pipe-filter, mvc, layer 등의 스타일 혼용 적용 가
  - 후보 아키텍처 도출: 배경도 및 각 관점별 다이어그램 작성, 아키텍처 명세서 기술
- 검증 및 승인
  - 아키텍처 평가: 아키텍처 요구 사항 만족도, 적합성, 품질 속성간 절충 관계 등 평가
  - 아키텍처 상세화(반복): 설계 방법 도출, 설계 패턴 고려
  - 아키텍처 승인: 이해 관계자들이 최종 승인

13. 아키텍처의 4+1 관점(1)

그림 5-19 장님 코끼리 만지는 격

14. 아키텍처의 4+1 관점(2)

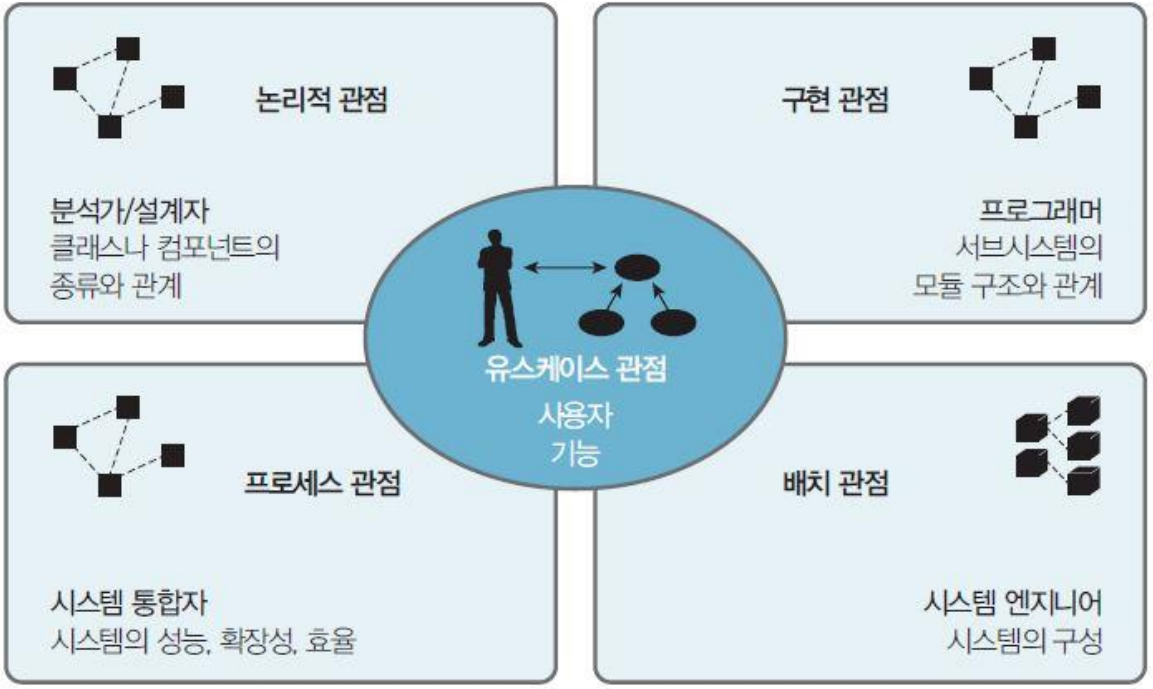

그림 5-20 UML 4+1 관점

15. 아키텍처의 4+1 관점(3)

- Usecase view
  - 시스템이 사용자에게 제공하는 기능에 관심
  - 다른 네 가지 관점에 사용되는 다이어그램의 근간
    - 정적 표현 : 유스케이스 다이어그램
    - 동적 표현 : 상태 다이어그램, 순차 다이어그램, 통신 다이어그램, 활동 다이어그램
- logical view(design view)
  - 클래스나 컴포넌트의 종류와 이들의 관계에 초점
    - 정적 표현 : 클래스 다이어그램, 객체 다이어그램
    - 동적 표현 : 상태 다이어그램, 순차 다이어그램, 통신 다이어그램, 활동 다이어그램

16. 아키텍처의 4+1 관점(4)

- implementation view
  - 소프트웨어 서브시스템의 모듈이 어떻게 구조화되어 있는가에 관심
    - 정적 표현 : 컴포넌트 다이어그램
    - 동적 표현 : 상태 다이어그램, 순차 다이어그램, 통신 다이어그램, 활동 다이어그램
- process view
  - 실제 구동 환경을 살펴봄으로써 논리적 관점과 같이 시스템 내부의 구조에 초점
  - 시스템의 동시성과 동기화에 관심
    - 동적 표현 : 상태 다이어그램, 순차 다이어그램, 협동 다이어그램, 활동 다이어그램
    - 시스템 구성 표현 : 컴포넌트 다이어그램, 배치 다이어그램

17. 아키텍처의 4+1 관점(5)

- deployment view
  - 시스템을 구성하는 처리 장치 간의 물리적인 배치에 초점
  - 서브시스템들이 물리적인 환경에서 어떻게 연관되어 실행되는지를 나타냄.
  - 시스템의 분산 구조와 실행할 때 컴포넌트들의 배치 상태를 나타냄.
    - 정적 표현 : 배치 다이어그램
    - 동적 표현 : 상태 다이어그램, 순차 다이어그램, 통신 다이어그램, 활동 다이어그램

18. 아키텍처 스타일(1)

- 음식 스타일
  - 한식, 일식, 중식
  - 음식 스타일에 따라 재료, 조리 방법, 담을 그릇 등 결정

그림 5-21 음식 스타일 결정

19. 아키텍처 스타일(2)

- 아키텍처 스타일에 따라
  - 구조, 규칙, 요소, 기법 등이 결정
  - 소프트웨어 특성, 전체 구조, 개발 방법을 알 수 있다.
- 좋은 소프트웨어 아키텍처 설계
  - 소프트웨어에 적합한 아키텍처 스타일을 선택하고 적용하고 통합하는 것
- 아키텍처 스타일을 사용한 설계의 장점
  - 개발 기간 단축, 고품질의 소프트웨어 생산
  - 수월한 의사소통
  - 용이한 유지보수
  - 검증된 아키텍처
  - 구축 전 시스템 특성에 대한 시뮬레이션 가능
  - 기존 시스템에 대한 빠른 이해

20. 아키텍처 스타일의 기능

- 소프트웨어 시스템의 구조를 체계적으로 구성하기 위해 기본 스키마를 제시
- 미리 정의된 서브시스템 제공
- 각 아키텍처 패턴 간의 책임 명시
- 패턴 간의 관계를 조직화하는 규칙, 가이드라인 제시
- 문제를 소프트웨어 모듈 단위로 분해하는 방법 제시
- 분해한 소프트웨어 모듈 단위가 상호작용하는 방법 제시

21. 아키텍처 모델

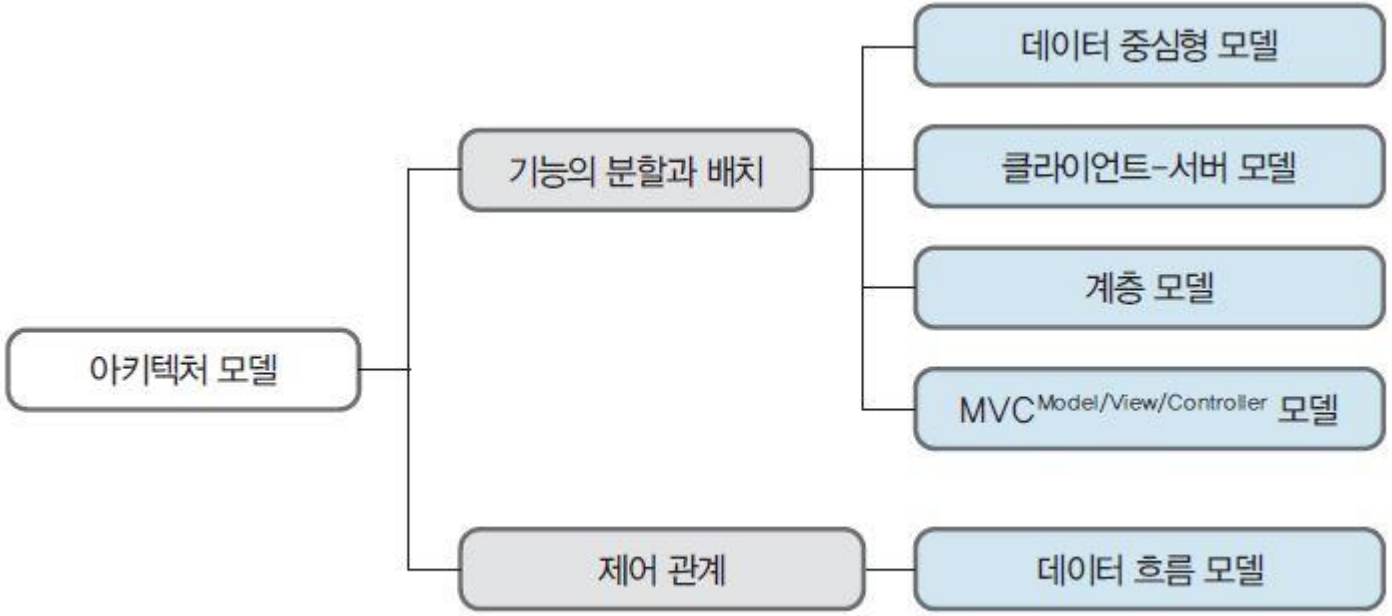

그림 5-22 **아키텍처 모델의 분류**

22. 데이터 중심형 모델(1)

- repository model
  - 특징: 주요 데이터가 repository에서 중앙 관리
  - 구성: repository와 여기에 접근하는 서브시스템
    - repository : 공동으로 활용하는 데이터 보관
    - 서브시스템: repository에 접근하여 정보를 저장, 검색, 변경하는 역할
  - 대량의 데이터를 공유하는 은행 업무 시스템에 매우 유용한 모델

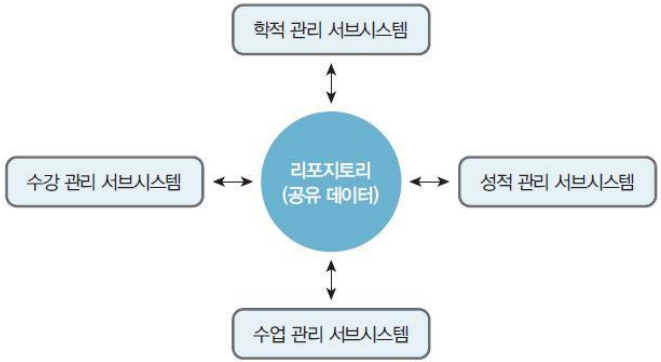

그림 5-23 데이터 중심형 모델

23. 데이터 중심형 모델(2)

- 장점
  - 데이터가 한군데에 모여 있기 때문에 데이터를 모순되지 않고 일관성 있게 관리 가능
  - 새로운 서브시스템의 추가 용이

- 단점
  - repository의 병목 현상 발생 가능
  - 서브시스템과 repository 사이의 강한 결합
    - ❖ repository 변경 시 서브시스템에 영향을 줌

24. client-server 모델

- Client-server 모델
  - 네트워크를 이용하는 분산 시스템 형태
  - 데이터와 처리 기능을 클라이언트와 서버에 분할하여 사용
  - 분산 아키텍처에 유용
    - ✓ 서버: 클라이언트(서브시스템)에 서비스 제공
    - ✓ 클라이언트: 서버가 제공하는 서비스를 요청(호출)하는 서브시스템

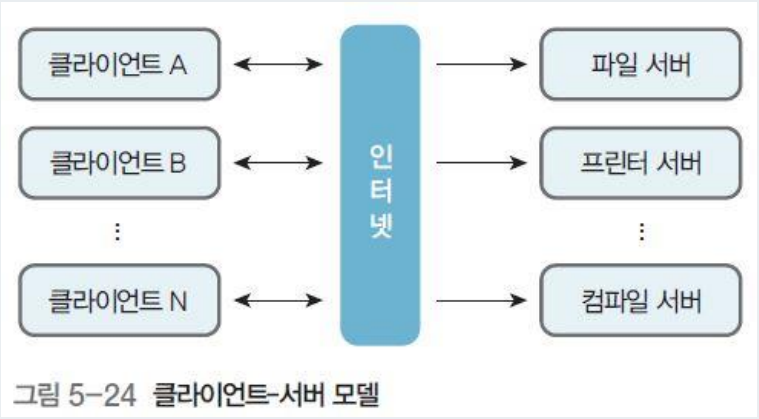

그림 5-24 클라이언트-서버 모델

25. 계층 모델

- layering 모델
  - 기능을 몇 개의 계층으로 나누어 배치
  - 구성: 하위 계층은 서버, 상위 계층은 클라이언트 역할

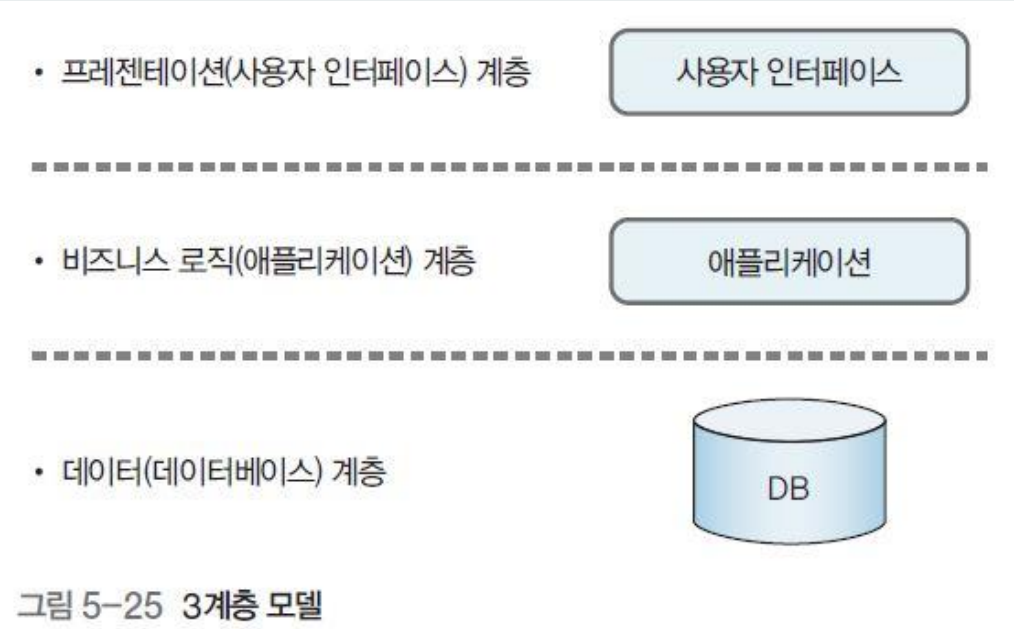

그림 5-25 3계층 모델

26. MVC 모델(1)

- Model/View/Controller 모델
  - 중앙 데이터 구조
  - 같은 모델의 서브시스템에 대하여 여러 뷰 서브시스템을 필요로 하는 시스템에 적합
  - 세 개의 서브시스템으로 분리하는 이유: 변경에 대한 영향을 덜 미치도록 하기 위해
    즉 UI부분이 자주 변경되더라도 모델 서브시스템에는 영향을 주지 않기 위해

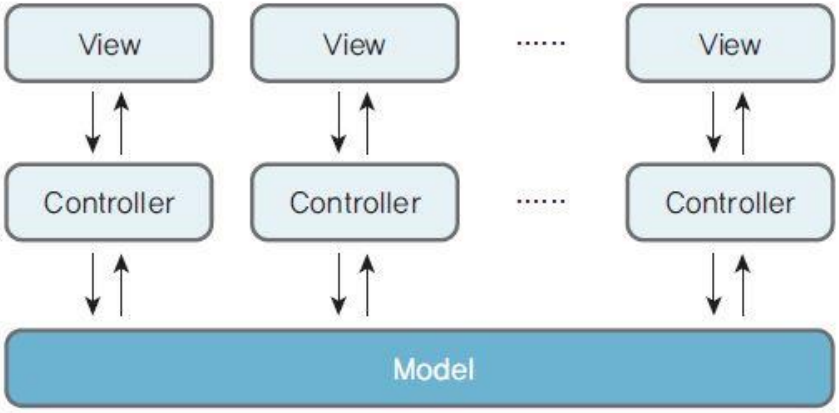

그림 5-26 MVC 모델

27. MVC 모델(2)

- Model 서브시스템
  - 뷰/제어 서브시스템과 독립되어 모든 데이터 상태와 로직을 처리
  - 특정 입·출력 방식에 영향을 받지 않고, 무언가의 호출에 응답만 함
- View 서브시스템
  - 사용자와 직접 대화가 이루어지는 부분으로 데이터를 사용자에게 보여주는 역할
- Controller 서브시스템
  - 뷰를 통한 사용자의 요청을 적절한 모델 쪽으로 넘겨주고, 모델로부터 받은 응답을 다시 뷰를 통해 사용자에게 돌려주는 역할

28. MVC 모델(3)

- 장점
  - 관심의 분리
  - 데이터를 화면에 표현(뷰)하는 디자인과 로직(모델)을 분리함으로써 느슨한 결합 가능
  - 구조 변경 요청 시 수정 용이
- 단점
  - 기본 기능 설계로 인한 클래스 수의 증가로 복잡도 증가
  - 속도가 중요한 프로젝트에 부적합

29. 데이터 흐름 모델

- Pipe and filter 구조
  - ✓ Filter: data stream을 한 개 이상 입력 받아 처리(변환)한 후 data stream 하나를 출력
  - ✓ pipe: filter를 거쳐 생성된 data stream 하나를 다른 filter의 입력에 연결

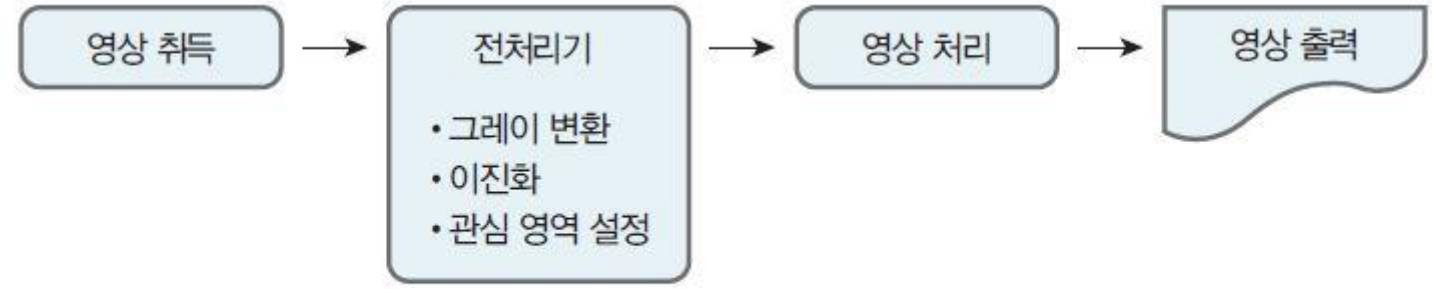

그림 5-27 데이터 흐름 모델의 예 : 이미지 프로세싱 과정

다음 시간

디자인 패턴

숭실사이버대학교의 강의콘텐츠는
저작권법에 의하여 보호를 받는바, 무단
전재, 배포, 전송, 대여 등을 금합니다.
*사용서체 : 나눔글꼴