Software Architecture

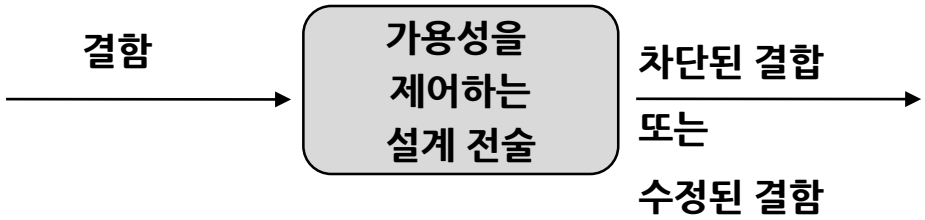
SW 아키텍처의 설계

품질 목표 달성

❖ 설계 전술

만일 고성능을 목표로 할 때 무엇을 통하여 원하는 품질속성을 달성할 수 있을까?

- 추구하는 품질이 달성될지 여부는 기본적인 설계 결정(설계전술)에 따름
- **품질속성의 응답을 제어하는 영향을 주는 설계 결정 사항**
- 설계전술의 집합은 아키텍처 전략
- 아키텍처 패턴은 설계전술을 일정 기준으로 묶은 패키지



❖ 가용성 설계 전술

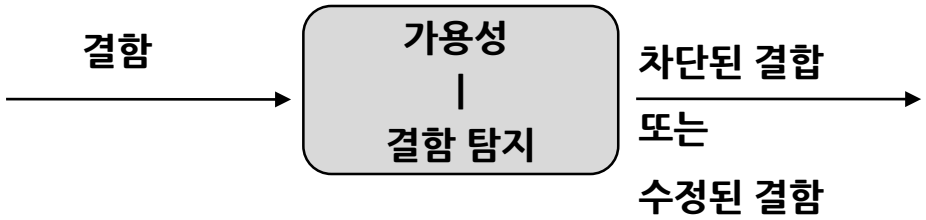
시스템이 명시된 서비스를 더 이상 제공하지 못한다면 실패가 발생

결함은 실패의 요인이 되는 잠재적 요소

- 가용성 유지 접근 방법 : 자동, 수동

  ➤ 결함 탐지 : 실패를 탐지하기 위한 것
  ➤ 결함 복구 : 실패 탐지 이후에 대한 것
  ➤ 결함 방지

가용성 설계 전술



❖ 결함 탐지의 설계 전술

- 결함을 찾는 가장 많이 사용되는 방식

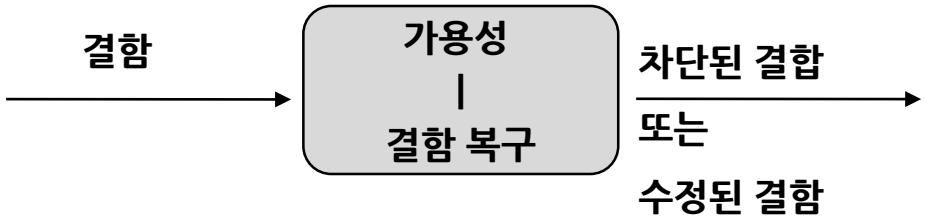
➤ 핑/에코(ping/echo)
- 컴포넌트가 신호를 보내고 상대 컴포넌트에서 정해진 시간에 응답이 오는지 관찰

➤ 생명주기신호(heartbeat)
- 컴포넌트가 주기적으로 생명주기 메세지를 보내면 다른 컴포넌트가 받음

➤ 예외 발생(exceptions)
- 누락, 정지, 타이밍, 응답 중 하나가 인식되는 상황
- 예외가 일어난 프로세스 안에서 예외처리기가 실행됨

가용성 설계 전술

❖ 결함 복구의 설계 전술



- 복구 준비하는 과정과 시스템을 수정하는 과정으로 구분됨

➤ 투표하기(voting)
➤ 활성 중복(active redundancy)
➤ 비활성 중복(passive redundancy)
➤ 예비(spare)
➤ 그림자 동작(shadow operation)
➤ 상태 재동기화(state resynchronization)
➤ 체크포인트/롤백(checkpoint/rollback)

가용성의 설계 전술 요약

❖ 결함 방지의 설계 전술

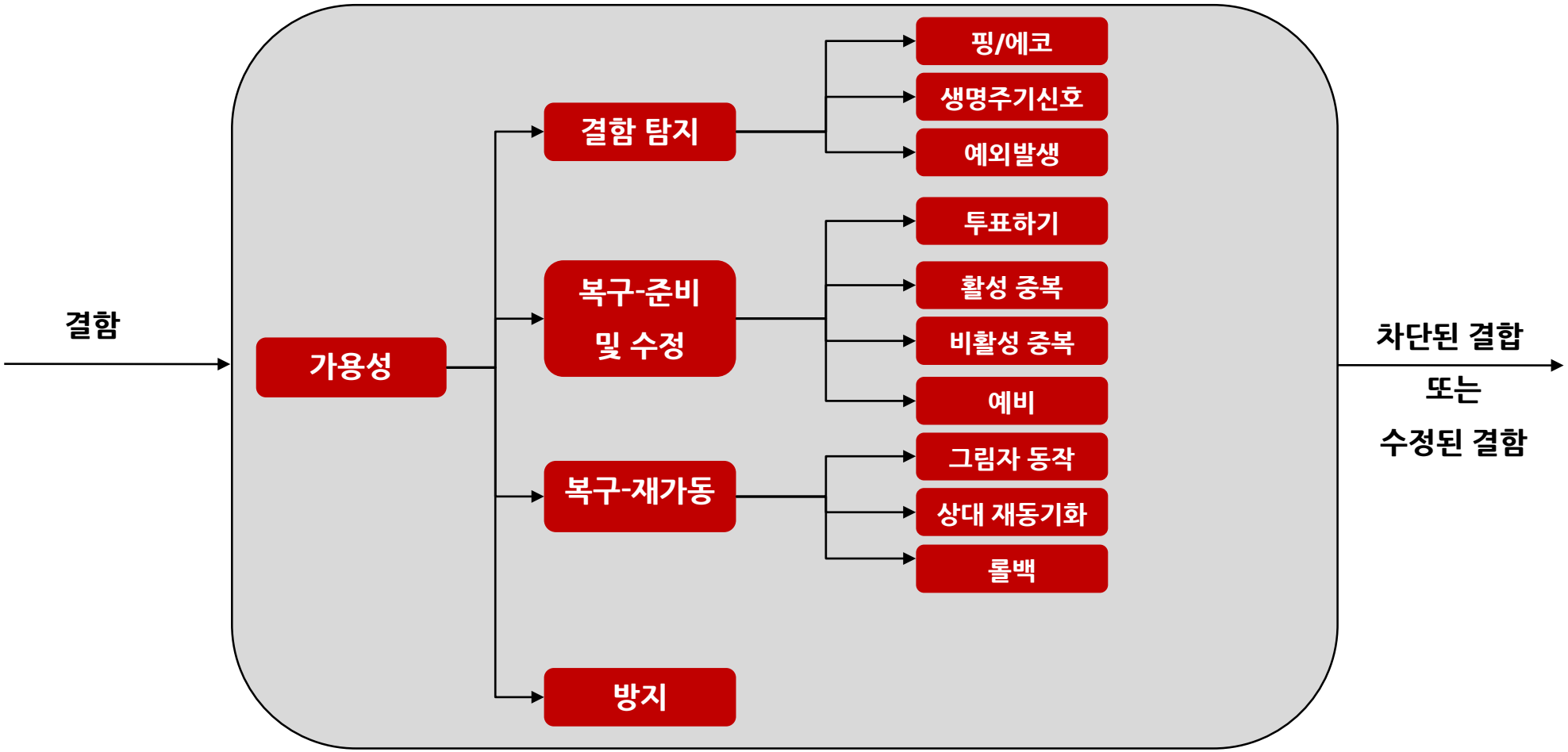
➤ 서비스 제거(removal from service)
- 실패의 예상시 실행 중인 특정 컴포넌트를 제거
- 메모리 누수를 막기 위해 컴포넌트를 재시작

➤ 트랜잭션(transaction)
- 일련의 순차적인 절차를 한번에 원상태로 돌릴 수 있도록 묶음

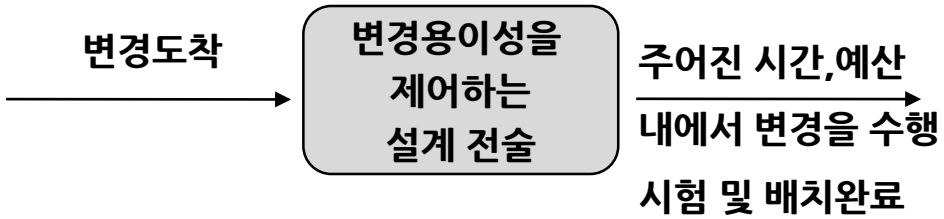
➤ 프로세스 감시(process monitor)
- 결함을 감지하면 중지된 프로세스를 삭제하고 프로세스의 신규 인스턴스를 생성

가용성 설계 전술

❖ 가용성의 설계 전술



❖ 변경용이성 설계 전술

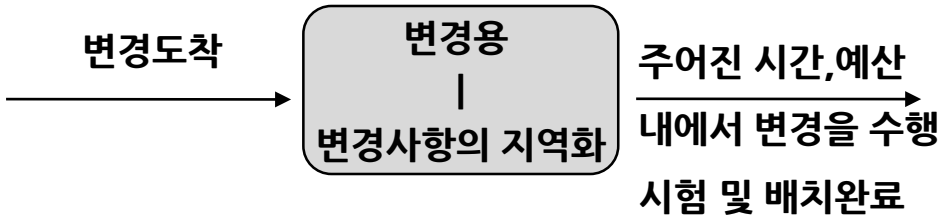


변경을 구현, 시험, 배치하는 데 소요되는 시간과 비용을 제어하는 것

➤ 변경사항의 지역화
- 변경에 의해 직접적으로 영향을 받는 모듈의 수를 줄임

➤ 파급효과의 방지
- 변경작업들을 지역화된 모듈로만 제한

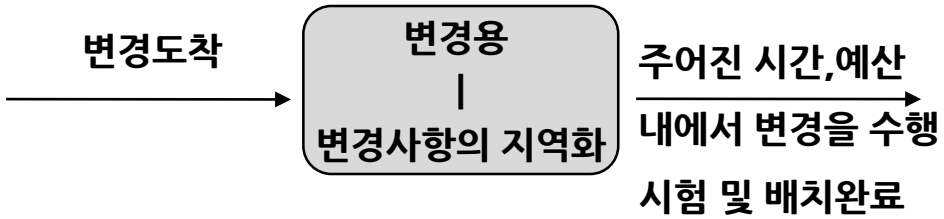
❖ 변경사항의 지역화



설계 단계에서 모듈의 담당 기능을 정하고 제한

- ➤ 의미적 응집력 유지
- ➤ 변경처리 예상
- ➤ 모듈의 일반화
- ➤ 변경의 폭 제한
- ➤ 추상 공통 서비스

❖ 파급효과의 방지

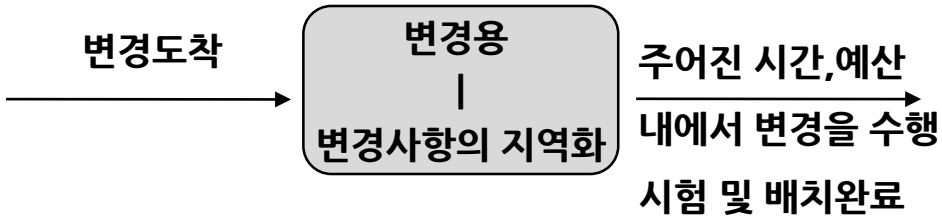


변경이 직접적으로 가해지지않은 모듈을 변경해야하는 경우

- ➤ 정보 은닉
- ➤ 기존 인터페이스 유지
- ➤ 통신경로 제한
- ➤ 중계자 사용

변경용이성 설계 전술

❖ 바인딩 시점의 연기

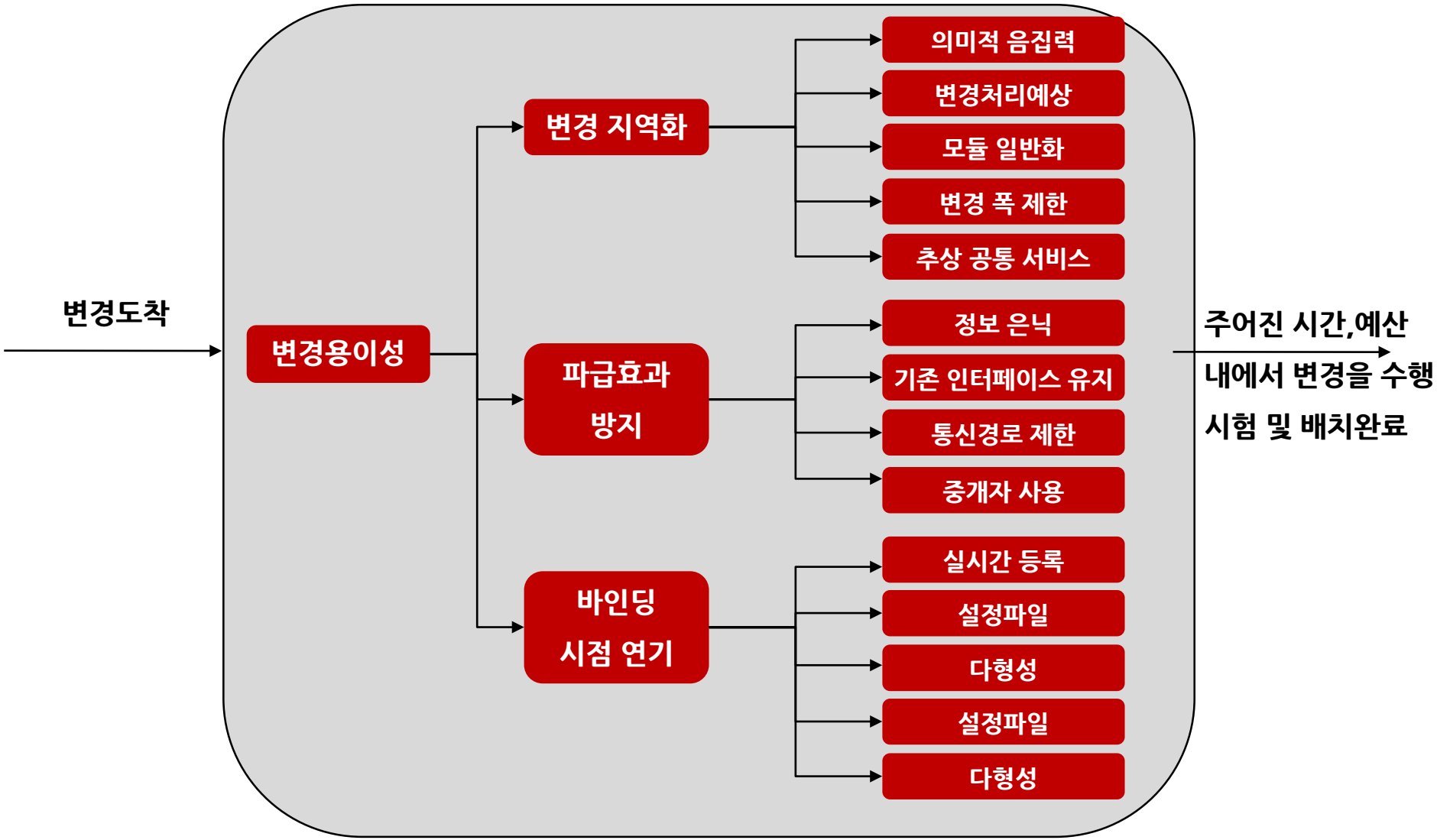


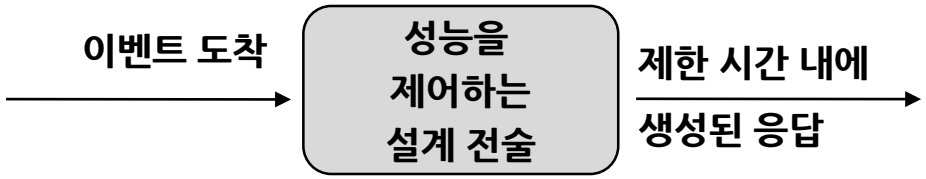
변경될 모듈의 수를 줄이는 것으로 만족될 수 없는 배치시점과 비개발자의 변경을 허용하는 경우

- ➤ 실시간 등록
- ➤ 설정 파일
- ➤ 다형성
- ➤ 컴포넌트 교체
- ➤ 정해진 프로토콜 준수

변경용이성 설계 전술

❖ 변경용이성의 설계 전술





❖ 성능 설계 전술

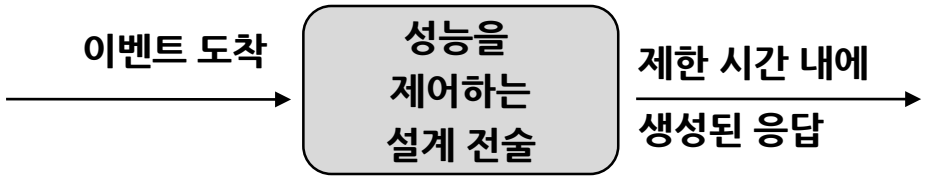
시스템에 도착한 이벤트에 대한 응답을 제약 시간 내에 만들어 내는 것

- **응답시간에 영향을 주는 기본 항목**

➤ 자원 소비
- CPU, 데이터 저장소, 네트워크 통신, 대역폭, 메모리 등
- 설계 시점에 정의한 객체 : 버퍼

➤ 자원 대기
- 자원에 대한 경쟁
- 자원 가용성
- 다른 연산에 대한 의존

설계 전술



❖ 성능 설계 전술

➤ 자원 요구
- 자원에 발생하는 이벤트 간의 시간
- 각 요청이 자원을 소비하는 양

➤ 자원 관리
- 병행성 적용
- 데이터 및 연산의 다중 사본 유지
- 가용 자원 증대

➤ 자원 조정
- 자원을 차지하려는 경쟁이 있는 곳에서는 반드시 자원(프로세스, 버퍼, 네트워크 등)을 스케쥴링해야함
- 각 자원이 사용될 시점의 특성을 파악하고 스케쥴링 전략을 수립

성능 설계 전술

❖ 자원 요구 설계 전술

이벤트 흐름을 처리하는 데 필요한 자원을 줄이는 것

➤ 연산 효율성 제고
- 사용되는 알고리즘을 개선

➤ 연산 과부하 축소
- RMI보다는 자바클래스 사용

처리되는 이벤트의 수를 줄이는 방법

➤ 이벤트 비율 관리

➤ 샘플링 빈도 조정

자원 사용을 제어

➤ 실행 시간 제한

➤ 큐 크기 제한

성능 설계 전술

❖ 자원 관리 설계 전술

➤ 병행성 적용
- 서로 다른 스레드에서 상이한 이벤트 흐름을 처리
- 스레드를 추가해 다른 동작을 처리

➤ 데이터 및 연산의 다중 사본 유지
- 중앙서버에서 모든 연산이 일어나는 경우 발생하는 자원 경쟁을 줄이는 것이 복제의 목적(캐싱)

➤ 가용 자원 증대
- 프로세서의 속도향상 및 추가, 메모리 추가, 고성능의 네트워크 등
- 자원을 늘리는 것은 대기시간을 줄여주는 확실한 설계전술

성능 설계 전술

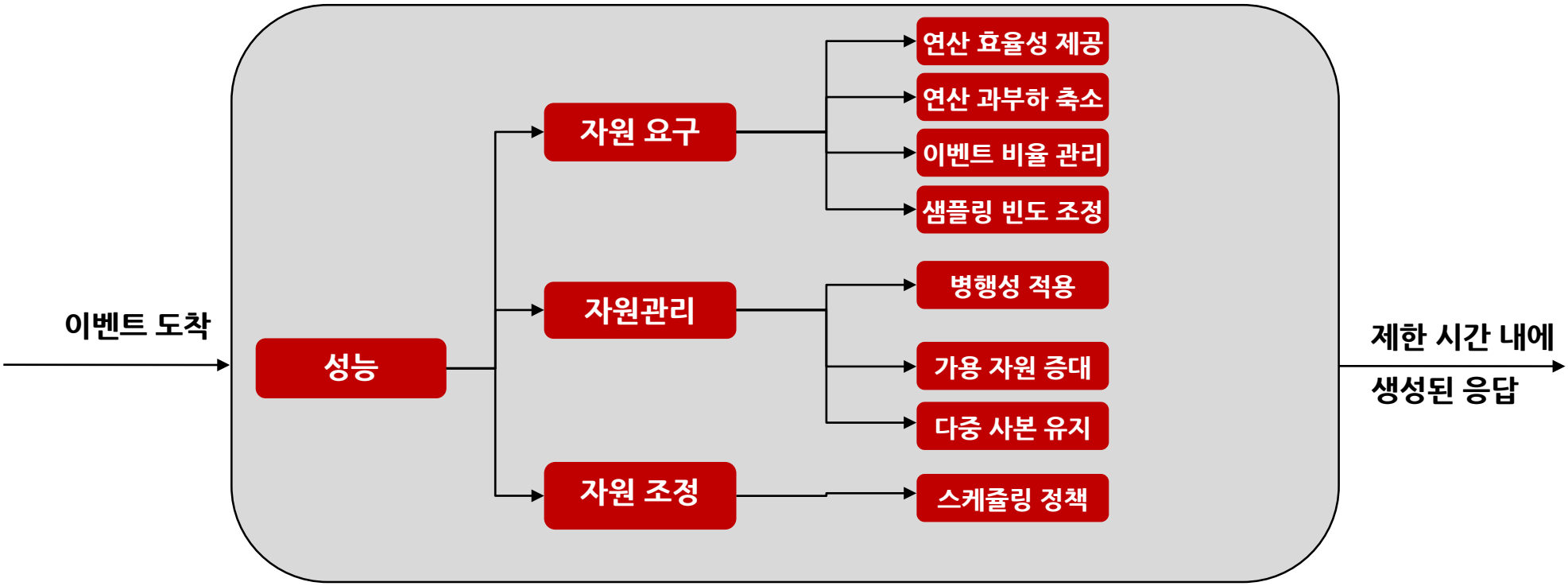
❖ 자원 조정 설계 전술

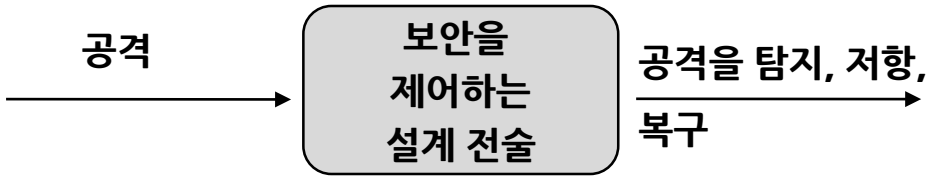
스케쥴링 정책

- ➤ 선입선출
- ➤ 고정 우선순위 스케쥴링
  - 의미 중요도
  - 기한 단조
  - 비율 단조
- ➤ 동적 우선순위 스케쥴링
  - 라운드 로빈
  - 최단 기한 우선
- ➤ 정적 스케쥴링
  - 순환 실행체제 스케쥴은 선점점과 자원 할당 순서를 오프라인에서 결정

성능 설계 전술

❖ 성능의 설계 전술





❖ 보안 설계 전술

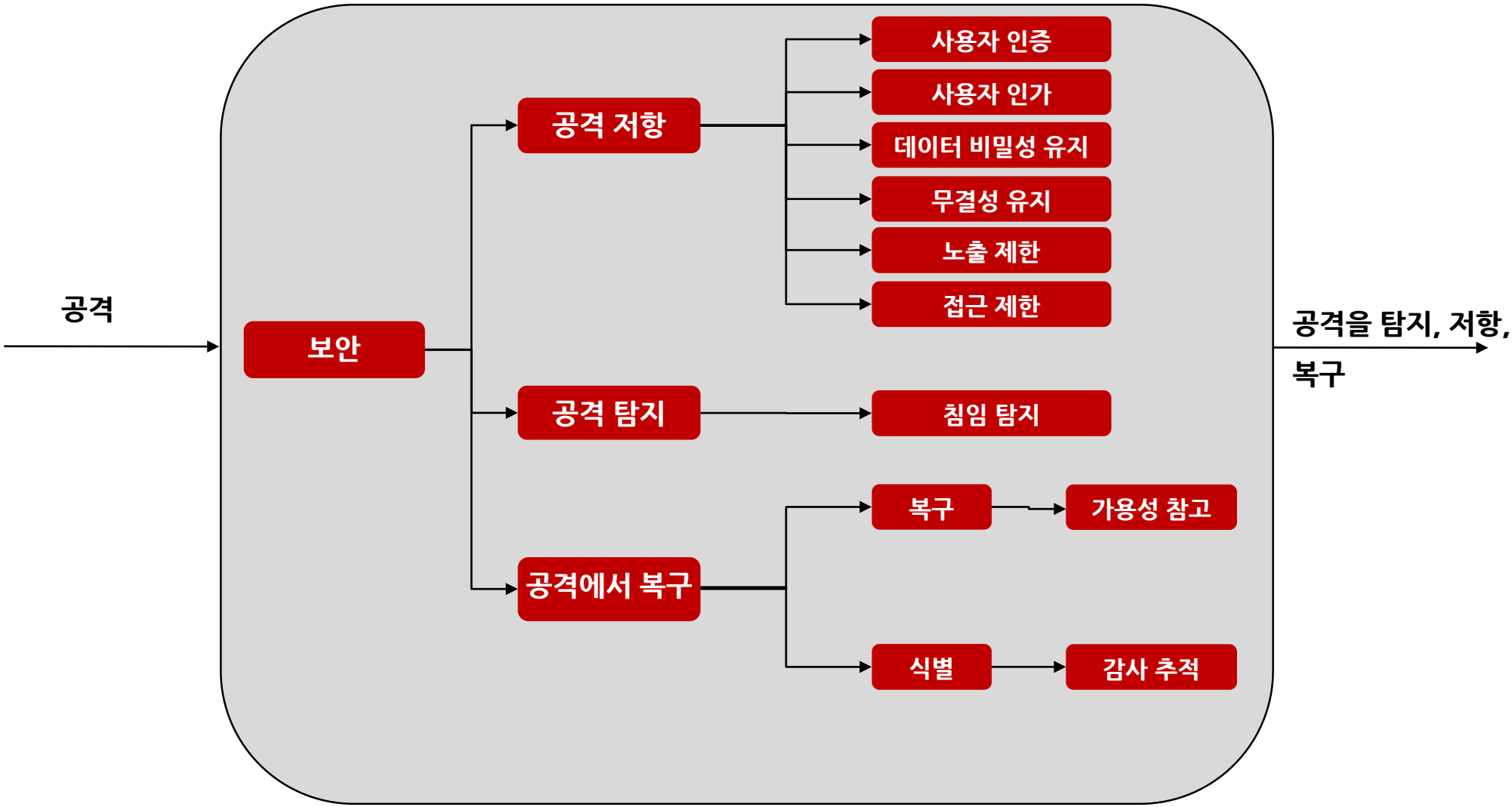
➤ 공격에 대한 저항
- 사용자 인증
- 사용자 인가
- 데이터 기밀성 유지
- 무결성 유지
- 노출 제한
- 접근 제한

➤ 공격 탐지
- 침입 탐지 시스템을 이용해 공격을 탐지

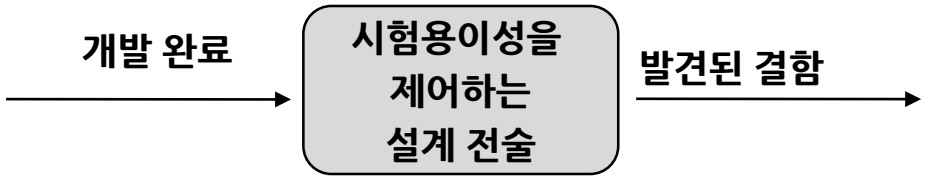
➤ 공격의 복구
- 복구 : 가용성 참고
- 식별 : 감사추적

보안 설계 전술

❖ 보안 설계 전술의 요약



설계 전술

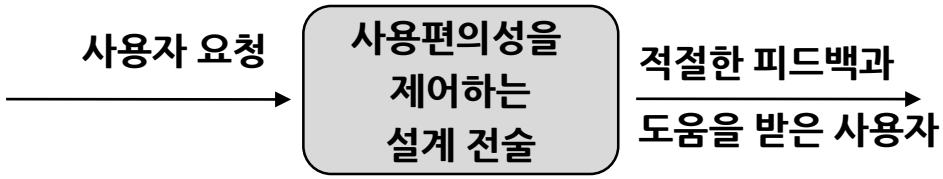


❖ 시험용이성 설계 전술

소프트웨어를 점진적으로 개발할때 증분마다 쉽게 시험을 수행하기 위한 것

➤ 입력/출력
- 기록/되풀이
  - 인터페이스를 통과하는 정보를 모으고 그 정보를 시험 입력으로 사용
- 구현에서 인터페이스를 분리
- 접근 경로/인터페이스의 특성화

➤ 내부 감시
- 내부 감시자(Build-in monitors)
  - 컴포넌트의 인터페이를 통해 접근할 수 있는 상태, 성능 부하, 용적, 보안 등의 정보를 유지



❖ 사용편의성 설계 전술

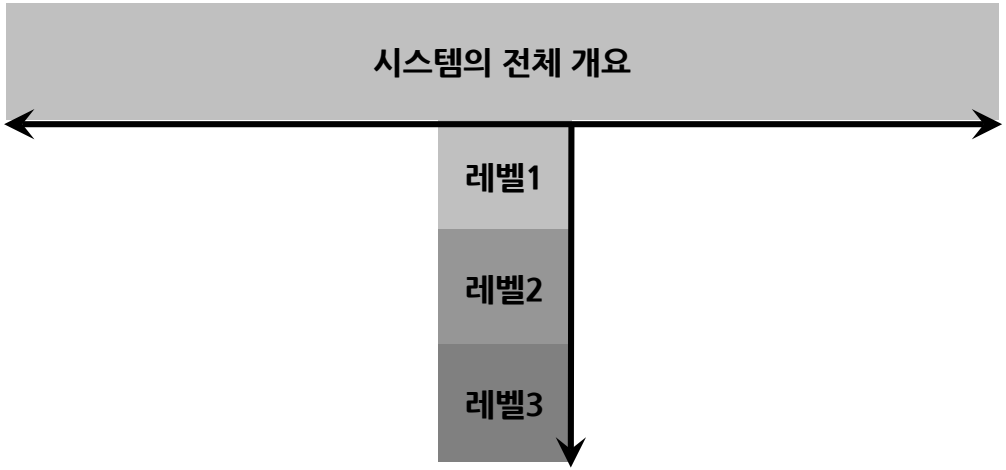
사용자가 자신이 의도한 작업을 쉽게 달성하는지 그리고 시스템은 사용자에게 어떤 도움을 주는지

➤ 사용자 인터페이스의 분리
- MVC(Mode-View-Control)
- PAC(Presentation-Abstraction-Control)

➤ 사용자 의도 지원
- 취소
- 되돌리기
- 통합

➤ 시스템 의도 지원
- 작업 모델 유지
- 사용자 모델 유지
- 시스템 모델 유지

성공적인 아키텍처 설계 원칙



❖ 성공적으로 아키텍처를 설계하기 위해서는 생각하는 방법이 필요

➤ 시스템의 전체적인 개요를 이해

➤ 시스템을 추상화시켜 시스템을 의미있는 단위로 분할

➤ 서브시스템이나 컴포넌트 등의 모듈로 그룹화함으로써 모듈화

➤ 분할된 모듈을 하나씩 나누어서 정복

- 정복해야 하는 하나의 모듈을 선택
- 한 모듈의 범위 안에서 하나의 레벨을 이해했다면 관심을 다른 모듈로 이동

아키텍처 설계 단계

❖ 아키텍처 설계의 사전 작업

➤ 초기 아키텍처 개요를 정의

- 사용자의 비기능적인 요구사항에서 품질 속성을 도출하고 품질 속성을 실현할 수 있는 전략을 수립하여 가장 실현 가능한 초기 아키텍처의 정의
  - 생성된 초기 아키텍처는 이후 아키텍처 설계를 위한 모든 과정의 기초가 됨

➤ 행위 분석

- 주요 작업은 유스케이스 분석(use case analysis)
  - 이 작업은 시퀀스 다이어그램으로 표현되는 유스케이스 실현을 통해 유스케이스 즉 시스템이 행하는 행위를 분석하고 설명하는 것
  - 이 과정 속에서 분석 클래스(analysis class)라고 하는 개념적인 소프트웨어 요소를 찾아내게 되며, 유스케이스 실현은 이들 개념적인 소프트웨어 요소들이 서로 상호작용하는 방법을 표현함으로써 유스케이스의 목적 즉, 사용자의 기능적인 요구사항을 어떻게 달성하는지를 보여줌
- 유스케이스 실현의 결과물은 이후 아키텍처 설계 단계의 활동과 작업에 입력물로 활용

아키텍처 설계 단계

❖ 초기 아키텍처 모델 정의

- 이후 아키텍처 설계를 위한 모든 과정의 기반
  - 분석클래스를 식별하는 기준이 되며, 비즈니스 컴포넌트를 식별하는 기준
  - 기술 유스케이스 실현과 프레임워크 설계에 초기 아키텍처가 반영
  - 배포 뷰를 구성하는 배포 다이어그램의 기반
  - 이후 아키텍처 설계 과정에서 초기에 수립된 아키텍처의 세부적인 사항이 변경될 수 있음
  - 따라서 초기 아키텍처 정의 작업에서는 아직 아키텍처가 확정되는 것은 아님

- 아키텍처 정의 절차
  1. 품질 속성 도출 및 우선 순위 부여
  2. 아키텍처 전략 수립
  3. 레퍼런스 아키텍처 선택
  4. 초기 아키텍처 모델 정의

애플리케이션 아키텍처 정의 단계

❖ 아키텍처 설계 과정

- 비즈니스 컴포넌트와 이들 사이의 상호작용이란 관점에서 시스템의 구조를 논리뷰(logical view)로 표현

- 비즈니스 컴포넌트에서 사용자 인터페이스와 관련된 부분을 분리하여 사용자 인터페이스 뷰(user interface view)로 표현하고, 논리적인 비즈니스 컴포넌트를 구현하는 물리적인 소프트웨어 모듈의 구조를 구현 뷰(implementation view)로 표현
- 이들 뷰 중 구현 뷰는 구현 단계의 구현 모델 구조화 활동에서 정의되며, 다른 두 개의 뷰는 아키텍처 정의 단계에서 정의
- 애플리케이션 아키텍처 설계는 행위 분석 활동의 유스케이스 실현의 결과를 활용하는 가장 중요한 활동

  ❶ 이 활동에서는 유스케이스 실현 과정에서 도출된 개념적인 소프트웨어 요소 즉, 분석클래스들 중에서 비즈니스 컴포넌트의 후보를 도출

  ❷ 다른 소프트웨어 요소와 이들 후보 비즈니스 컴포넌트(candidate business component) 사이의 상호작용 즉, 메시지 흐름을 분석하여 후보 비즈니스 컴포넌트에 책임(responsibility)을 할당

  ❸ 후보 비즈니스 컴포넌트 중에서 비즈니스 컴포넌트(business component)를 도출하고, 비즈니스 인터페이스를 정의하고 설계

  ❹ 애플리케이션 아키텍처의 논리 뷰(logical view)에 이들 비즈니스 컴포넌트와 함께, 비즈니스 인터페이스를 통한 비즈니스 컴포넌트들의 상호작용과 관계를 클래스 다이어그램으로 표현

아키텍처 정의 단계

❖ 아키텍처 설계 과정 개요

- 애플리케이션 아키텍처 설계
  - 일련의 활동 및 작업들과 병행하여 사용자 인터페이스 뷰(user interface view)를 정의하기 위한 사용자 인터페이스 모델 생성 작업이 실행
    - 이 작업은 유스케이스 스토리보드(use case storyboard)를 생성하여 사용자 인터페이스 관점에서 유스케이스 실현을 실현함으로써 사용자 인터페이스 요소들이 어떻게 상호 작용하는지를 기술
    - 그 결과로써 사용자 인터페이스 클래스 다이어그램(user interface class diagram)과 화면 이동 맵(screen navigation map)을 생성
    - 이들 결과는 사용자 인터페이스 뷰(user interface view)에 표현

- 기술 아키텍처 설계
  - 사용자의 비기능적인 요구사항을 실현하는 방법에 초점을 맞춤
  - 전적으로 기술적인 요구사항을 해결하는데 집중
  - 기술적인 요구사항 해결
    - ❶ 기술 유스케이스(technical use case)를 정의
    - ❷ 기술 유스케이스 실현(technical use case realization)
    - ❸ 프레임워크(framework)에 참여하게 될 클래스를 도출
    - ❹ 프레임워크에 참여하는 클래스를 설계하고 구현
    - ❺ 프레임워크 정의
  - 기술 유스케이스 뷰를 포함(use case diagram, class diagram)
  - 배포 뷰를 포함(deployment diagram)

아키텍처 정의 단계

❖ 아키텍처 설계 과정 개요

- 데이터 아키텍처 설계
  - 행위 분석 활동의 유스케이스 실현의 결과를 활용하는 두 번째 활동

❶ 유스케이스 실현에 참여하는 분석클래스 사이에 교환되는 데이터를 분석하여 비즈니스 객체(business object)를 도출하고, 비즈니스 객체 모델(business object model)을 생성
❷ 이들 비즈니스 객체 중에서 지속성 처리를 요구하는 객체를 대상으로 정규화(normalization) 과정을 수행하여 논리 데이터 모델(logical data model)을 구성
❸ 논리 데이터 모델을 대상으로 선택된 데이터베이스에 적합한 비정규화(denormalization) 등의 과정을 수행하여 물리 데이터 모델(physical data model)을 생성
❹ 설계 단계에서 데이터 액세스 컴포넌트(data access component)의 인터페이스 실현(interface realization)에 참여하는 설계 요소(design element) 사이에 전달되는 데이터를 분석하여 저장 프로시저(stored procedure)를 설계

  - 데이터 논리 모델과 물리 모델은 E-R(entity -relationship) 모델로 정의되어 데이터 뷰(data view)에 포함

아키텍처 정의 단계

❖ 아키텍처 설계 및 프로토타이핑

- 아키텍처 설계가 완료되면 해당 아키텍처를 기반으로 설계 단계에서 설계자가 템플릿으로 사용할 설계 메커니즘과 구현 단계에서 개발자가 사용하게 될 구현 메커니즘을 정의

- 여기에서 정의되는 설계 메커니즘과 구현 메커니즘은 각각 설계자와 개발자들이 자신이 작업을 수행하는데 있어서 제약사항으로 작용

- 아키텍처 정의 단계에서 마지막 작업은 아키텍처 프로토타이핑

  - 아키텍처가 견고한지의 여부를 검증
    - 가장 위험요소가 많고 우선 순위가 높은 유스케이스를 선택하여 실제로 아키텍처와 설계 및 구현 메커니즘에 따라 설계, 구현, 테스트 과정을 실행

  - 이 작업의 결과는 다시 아키텍처 설계에 반영되며, 최종적으로 견고함이 검증된 아키텍처를 설계자와 개발자에게 제시

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

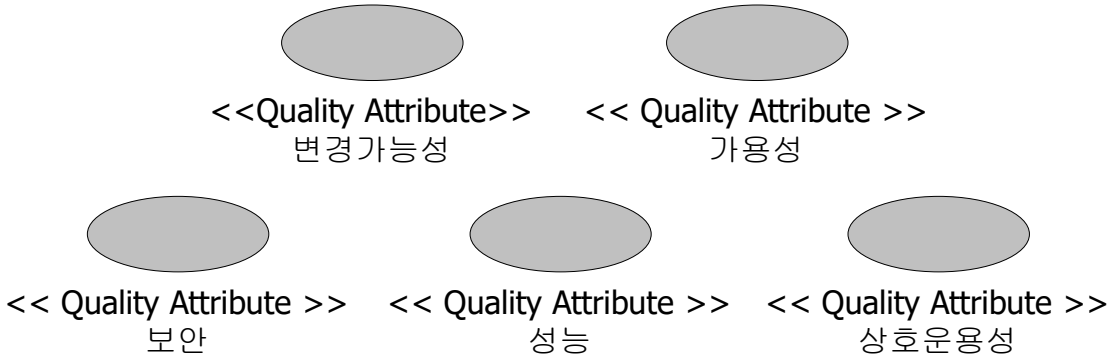
- ATAM 기법을 활용한 아키텍처 정의
  - Evaluating Software Architecture
    - ATAM(Architecture Tradeoff Analysis Method) 아키텍처 평가 기법을 설명
    - 모두 9단계의 과정으로 아키텍처를 평가하는 방법을 설명

1) 아키텍처 접근 방법(아키텍처 스타일) 식별
   - 후보 아키텍처 스타일을 식별하고 나열한다.
2) 품질 속성 유틸리티 트리(quality attribute utility tree) 생성
   - 품질 속성 시나리오를 도출하고 우선 순위를 결정한다.
3) 아키텍처 접근 방법(아키텍처 스타일) 분석
   - 품질 속성을 기준으로 후보 아키텍처를 평가한다.
4) 아키텍처 선정
   - 아키텍처를 선택한다.

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 품질 속성 도출 및 우선 순위 부여
  - 초기 아키텍처를 정의하기 위해 가장 처음 해야 할 일은 사용자의 비기능적인 요구사항으로부터 품질 속성을 도출하고, 이들 품질 속성에 대하여 우선 순위를 부여하는 것
  - 우선 순위는 다음과 같은 두 가지 관점을 고려하여 결정
    - 1. 시스템 성공에 있어서의 각 품질 속성의 중요성
    - 2. 아키텍트가 예상하는 품질 속성 실현 난이도
  - 품질 속성



- 도출된 품질 속성은 유스케이스기술서의 비기능적인 요구사항을 대체하여 정의

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 품질 속성 도출 및 우선 순위 부여

품질 속성	비기능 요구사항	우선순위
가용성	온라인 서점 관리 시스템은 실패없이 365일, 24시간 멈추지 않고 가동 (단, 시스템 정기점검 시간은 제외)되어야 하며, 시스템의 문제가 발생하는 경우 사용자에게 이를 통지하고 적절하게 대처해야 한다.	하
성능	온라인 서점 관리 시스템은 사용자 요청을 3초 안에 처리하여 결과를 제공해야 한다. 또, 동시 사용자 1000명을 지원해야 하므로 초당 300 이상의 트랜잭션을 처리해야 한다	중
보안	불특정 다수의 사용자가 이용하는 도서 쇼핑몰이지만 회원 정보의 유출이 되지 않도록 시스템이 보호되어야 한다.	중
	시스템의 접근 가능한 인증된 사용자라 할지라도, 그 수준에 따라서 사용할 수 있는 시스템의 기능에 차등을 두어야 한다. 예를 들어, 회원 고객과 비회원 고객은 인증된 사용할지라도 시스템으로부터 제공받 는 서비스가 다르다.	중

초기 아키텍처 개요 정의

품질 속성	비기능 요구사항	우선순위
변경가능성 (Modifiability)	온라인 서점의 배송 요청, 추적 서비스는 배송 업체에 따라 다를 수 있다. 따라서, 본 온라인 서점 관리 시스템은 향후 배송 관련 서비스의 손쉬운 변경을 허용할 수 있어야 한다.	상
	본 온라인 서점 관리 시스템은 우선적으로 온라인 서점의 판매업무를 위해서 설계 구현되지만, 향후 다양한 테마나 콘텐츠를 제공하는 시스템으로 확장될 수 있다. 따라서, 향후 시스템이 확장되더라도 별도의 성능 저하가 있어서는 안된다.	상
	온라인 서점 관리 시스템을 구성하는 각각의 비즈니스 컴포넌트는 별도의 코드 수정 없이 다른 컴포넌트로 교체하여도 정상적으로 동작하도록 한다. 단, 이때 교체되는 비즈니스 컴포넌트는 동일한 기능을 제공해야 하며, 기능 명세의 형식이 동일해야 한다. 그러나, 비즈니스 컴포넌트가 아닌 애플리케이션 컴포넌트의 경우 비즈니스 컴포넌트의 교체에 따라서 약간의 내용 수정을 허용하도록 한다.	상
사용성 (Usability)	온라인 서점은 인터넷 환경을 기반으로 구축되어야 하며, 누구나 웹 브라우저를 통해 온라인 서점에 접근할 수 있어야 한다.	상
	온라인 서점은 불특정 다수의 구매자가 손쉽게 사용할 수 있도록 사용이 쉽고 구성이 복잡하지 않아야 한다	중
상호운용성 (Interoperability)	온라인 서점은 아마존 도서 목록 서비스와 배송 업체의 배송 서비스를 사용하며 이들 서비스 사이의 통신은 웹 서비스를 통해 이루어진다.	상

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 아키텍처 기본 전략 수립
  - 애플리케이션 아키텍처뿐만 아니라 기술 아키텍처와 하드웨어 구성, 구현 등을 종합적으로 고려하며 아키텍처 스타일이나 아키텍처 패턴을 참조

- 유의해야 할 점
  - 도출된 모든 품질 속성 요구사항을 만족시킬 수 있는 아키텍처 전략을 수립하기는 거의 불가능
  - 품질 속성의 우선 순위에 따라서 아키텍처 전략에 가중치를 두어야 할 필요
  - 또한, 여기에서 선택한 아키텍처 전략은 아키텍처를 설계하는 과정 속에서 변경될 수 있으므로, 여기에서 완전히 확정된 아키텍처 전략을 수립해야 할 필요는 없음

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 아키텍처 기본 전략 수립

품질 속성	비기능 요구사항	설계 전술
변경 가능성	애플리케이션 아키텍처	레이어(layered) 아키텍처 스타일
	기술 아키텍처	미들웨어(EJB 컨테이너) 도입
		논리적인 레이어를 물리적인 레이어에 분산 배포
가용성	기술 아키텍처	클러스터링(clustering)
		트랜잭션(transaction)
	구현	예외처리(exception handling)
보안	기술 아키텍처	폼 인증(Forms authentication)
		권한(authorization)
		암복호화(encryption-decryption)
성능	기술 아키텍처	동적 로드 밸런싱(dynamic load balancing)
	하드웨어	CPU, 메모리 등 리소스 확장
상호운영성	기술 아키텍처	웹 서비스(web service)

❖ 초기 아키텍처 모델 정의

- 아키텍처 기본 전략 수립

  - 우선 순위가 가장 높은 변경 가능성 품질 속성의 요구사항

    ❶ '배송 서비스를 손 쉽게 변경' 할 수 있게 해달라

    ❷ '성능의 변화없이 시스템의 기능을 확장' 할 수 있게 해달라

    ❸ '컴포넌트를 손쉽게 대체' 할 수 있게 해달라

  - 엔터프라이즈 애플리케이션 시스템에서 논리적인 레이어(layer)는 물리적인 티어(tier)에 분산 배포되어야 할 필요가 있다.

초기 아키텍처 개요 정의

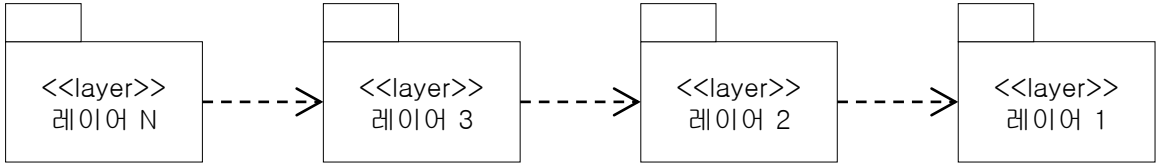
❖ 초기 아키텍처 모델 정의

- 레이어 아키텍처 스타일
  - 일반적으로 엔터프라이즈 시스템은 여러 추상화 단계를 갖는 많은 컴포넌트로 구성된다.
    - 따라서 엔터프라이즈 시스템에 요구되는 유지보수성(maintainability), 재사용성(reusability), 확장성(scalability), 강건성(robustness), 보안(security) 등의 요구사항을 지원하도록 이들 컴포넌트들을 구조화할 필요가 있다.
  - 이러한 목적으로 사용되는 것이 레이어 아키텍처 스타일(layered architecture style)이다.
  - 레이어 아키텍처 스타일은 다음과 같은 특징을 갖는다.
    - 시스템의 일부분으로 변경을 지역화하여 시스템의 다른 부분에 대한 영향을 최소화함으로써, 애플리케이션의 관리를 쉽게 하여 전반적인 애플리케이션의 유연성(flexibility)을 향상시킨다.
    - 컴포넌트 사이에 관심을 분리함으로써 유연성(flexibility), 유지보수성(maintainability), 확장성(scalability)을 향상시킨다.
    - 컴포넌트를 여러 애플리케이션에 재사용할 수 있다.
    - 다른 팀과의 종속성을 최소화한 상태로 각 팀은 자신의 작업을 수행할 수 있으며, 각 팀은 자신의 능력에 맞는 분야의 작업을 효율적으로 진행할 수 있다.
    - 각 컴포넌트는 내부적으로 강한 응집력(cohesion)을 가져야 한다.
    - 컴포넌트 사이에는 느슨한 결합도(loosely-coupled)를 가져야 한다.
    - 다양한 컴포넌트들은 독립적으로 배포되며 관리되어야 한다.
    - 너무 많은 레이어를 사용하면 성능에 부정적인 영향을 미치게 된다.

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 레이어 아키텍처 스타일
  - 컴포넌트는 레이어 안에 놓이게 되며, 레이어는 컴포넌트의 추상화 레벨을 나타낸다.
  - 일반적으로 같은 레이어 안에 있는 컴포넌트는 거의 같은 추상화 레벨을 가져야 하며, 다른 레이어의 컴포넌트 사이에는 느슨한 결합도(loosely-coupled)를 가져야 한다.
  - 레이어는 가장 낮은 추상화 레벨을 갖는 레이어 1에서부터 가장 높은 추상화 레벨을 갖는 레이어 N까지로 구성된다.
  - 레이어 구조



  - 한 레이어에 포함된 컴포넌트는 같은 레이어에 있거나 하위 레이어에 있는 컴포넌트와 서로 상호작용을 할 수 있으며, 이것은 다른 레이어에 있는 컴포넌트 사이의 종속성을 감소시켜주는 이점을 제공한다.
  - 레이어 방식
    - 엄격한 레이어(strict layered) 방식
      ≫ 같은 레벨의 레이어 또는 바로 하위 레벨의 레이어하고만 커뮤니케이션을 할 수 있도록 제한한다.
    - 유연한 레이어(flexible layering) 방식
      ≫ 같은 레벨의 레이어는 물론, 하위 레벨의 어떤 레이어하고도 커뮤니케이션이 가능하다.
      ≫ 상위 레벨의 레이어들에게 영향을 주지 않고 하위 레벨의 레이어를 변경시키는 일이 어려우므로 관리에 문제를 노출하기도 한다.

❖ 초기 아키텍처 모델 정의

- 레이어 아키텍처 스타일
  - 상호작용 방식
    - 하향식(top-down)
      - ≫ 시스템 외부에서는 가장 상위에 있는 레이어와 상호작용을 한다.
      - ≫ 최상위 레벨의 레이어는 하위 레벨의 레이어가 제공하는 서비스를 사용하고, 다시 각 하위 레벨의 레이어는 그 밑에 있는 하위 레벨의 레이어가 제공하는 서비스를 사용한다.
    - 상향식(bottom-up)
      - ≫ 하위 레이어는 상위 레이어를 직접 호출하지 못한다.
      - ≫ 하위 레이어는 이벤트(event), 콜백(callback), 위임(delegate) 등을 통해서 상위 레이어와 커뮤니케이션을 해야 한다.
      - ≫ 상향식 모드는 하위 레이어가 상위 레이어에 대하여 종속적인 상태가 되게 하므로 레이어 아키텍처를 사용하는 이점을 그만큼 반감시키게 된다.
      - ≫ 따라서 가능한 한 상향식 모드를 사용하지 않는 것이 바람직하다.
  - 논리적인 레이어(layer)는 물리적인 티어(tier)에 분산 배포된다.
  - 엔터프라이즈 애플리케이션 시스템을 구성하는 각 서버나 클라이언트 컴퓨터 시스템은 물리적인 티어로 구조화된다.
    - 따라서 각 티어는 하나 이상의 컴퓨터로 구성된다.
  - 같은 티어에 속해 있는 컴퓨터 시스템은 같은 특성을 공유한다.

레이어 아키텍처의 설계

❖ 레이어 아키텍처 설계 과정

❶ 작업을 레이어로 그룹화하기 위한 추상화 기준(abstraction criterion)을 정의한다. 이 기준은 플랫폼과는 직접 관련이 없는 개념적이며, 일반적으로 다음과 같은 기준으로 레이어를 나눌 수 있다.

사용자 인터페이스 요소
특정한 애플리케이션 모듈
공통 서비스 레벨
운영체제 인터페이스 레벨
운영체제
하드웨어

❷ 추상화 기준에 따라 추상화 레벨의 수를 결정한다. 각 추상화 레벨은 하나의 레이어에 대응한다. 때로는 추상화 레벨과 레이어의 매핑이 분명하지 않을 수도 있다. 또한, 너무 많은 레이어를 갖는 것은 불필요한 부하를 가져올 수 있다. 그러나 너무 적은 레이어는 빈약한 구조를 야기시킬 수 있다.

❸ 레이어에 이름을 부여하고 각 레이어에 작업을 할당한다. 최상위 레이어의 작업은 클라이언트에 의해 사용되는 것과 같은 전반적인 시스템 작업이다. 모든 다른 레이어의 작업은 상위 레이어의 헬퍼(helper)로서 역할을 한다.

❖ 레이어 아키텍처 설계 과정

❹ 서비스를 명시한다. 가장 중요한 구현 원리는 레이어가 서로 엄격하게 분리되어야 한다는 것이다. 어떤 컴포넌트도 여러 레이어에 걸쳐 있으면 안된다. 그러나 레이어 사이에 공유하는 모듈은 엄격한 레이어링의 원리를 완화시켜 적용할 필요가 있다. 또한, 하위 레이어 보다는 상위 레이어에 더 많은 서비스를 두는 것이 바람직하다. 이것을 '재사용성의 역 피라미드(inverted pyramid of reuse)라고 부른다.

❺ 레이어를 정제한다. 1에서 4단계까지를 반복한다. 일반적으로 레이어와 서비스를 고려하기 전에 추상화 기준을 정확하게 정의하는 것은 불가능하다. 반면에 처음에 컴포넌트와 서비스를 정의하고 나중에 이들의 사용 관계에 따라서 레이어 구조를 부여하는 것도 잘못될 가능성이 많다. 가장 좋은 방법은 자연스럽고 안정된 레이어 구조를 가질 때까지 처음 네 개의 단계를 여러 번 반복하는 것이다.

❻ 각 레이어에 대하여 인터페이스를 명시한다. 가능하다면 하위 레이어는 상위 레이어에 대하여 '블랙박스(black box)'가 되도록 인터페이스를 설계한다. 이 경우, 하위 레이어의 모든 서비스를 제공하는 인터페이스를 정의하고 퍼사드(façade) 객체에 이 인터페이스를 캡슐화한다. 효율성을 위해 필요한 경우 또는, 다른 레이어의 내부 구조에 접근해야만 하는 경우에는 예외적으로 '화이트박스(white box)' 또는 '그레이박스(gray box)' 방식을 허용할 수 있다. 그레이박스는 상위 레이어에서 하위 레이어가 몇 개의 컴포넌트로 구성되어 있다는 사실을 알고 있으며, 이들 컴포넌트에 각각 접근할 수 있지만 이들 컴포넌트의 내부 구조를 볼 수는 없게 한다.

❼ 각 레이어를 구조화한다. 각 레이어 내부는 무질서한 경우가 많다. 따라서 각 레이어가 복잡할 때에는 관련된 컴포넌트로 분할되어야 한다.

❖ 레이어 아키텍처 설계 과정

❽ 인접 레이어 사이에 커뮤니케이션을 명시한다. 레이어 사이의 커뮤니케이션에서 가장 자주 사용되는 메커니즘은 푸시 모델(push model)이다. 이 모델에서는 레이어 J가 레이어 J-1의 서비스를 호출할 때 필요한 정보가 서비스 호출의 일부로서 넘겨진다. 그 반대가 풀 모델(pull model)이며, 하위 레이어가 상위 레이어에서 필요한 정보를 가져올 때 발생한다. 그러나 풀 모델은 인접한 상위 레이어 사이에 추가적인 종속성을 가져올 수 있으므로 가능한 한 피하는 것이 좋으며, 꼭 필요하다면 콜백(callback) 등의 메커니즘을 사용하는 것이 바람직하다.

❾ 인접 레이어를 분리(decouple)한다.

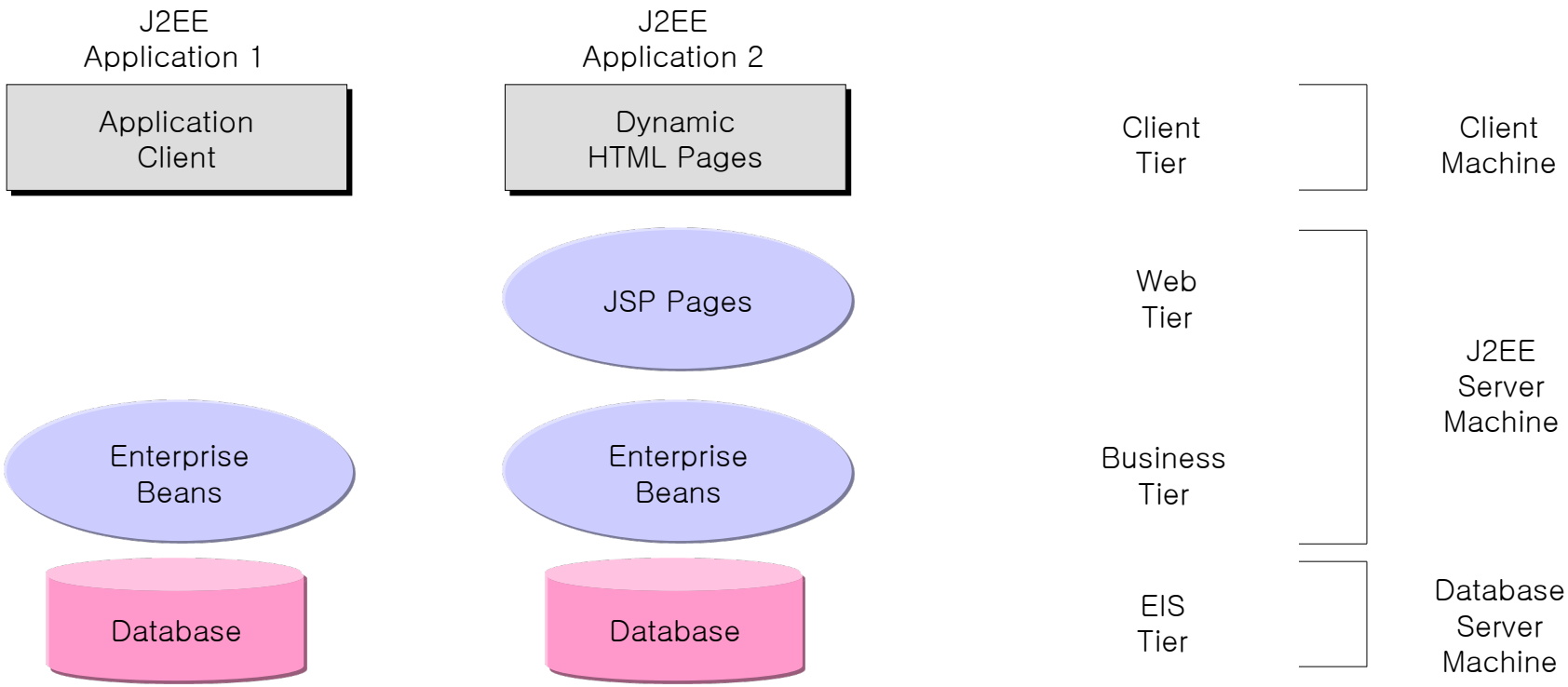
단방향 결합(one-way coupling) - 하향식 모드에 적절하다. 일반적으로 상위 레이어는 하위 레이어를 알 수는 있지만 하위 레이어는 사용자인 상위 레이어를 알지 못한다. 상향식 모드에서는 콜백(callback)을 사용하여 하향식 단방향 결합을 유지할 수 있다. 상위 레이어는 하위 레이어에 콜백 함수를 등록하고, 하위 레이어는 이벤트가 발생할 때 등록된 콜백 함수를 호출한다.

❿ 에러 처리 전략을 설계한다. 하위 레이어에서 에러가 발생할 때 자신이 처리하거나 다음 상위 레이어로 넘겨준다. 상위 레이어로 에러를 넘겨줄 때 하위 레이어는 상위 레이어가 이해할 수 있는 형태로 에러를 변형시켜야 한다. 가능한 한 하위 레이어에서 에러를 처리하는것이 바람직하다.

초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 레퍼런스 아키텍처 선택
  - 도출된 품질 속성과 아키텍처 전략을 만족하는 레퍼런스 아키텍처(reference architecture)를 선택하여 초기 아키텍처 개요의 골격으로 삼는다.
  - J2EE에서는 J2EE 애플리케이션 시스템을 위한 레퍼런스 아키텍처를 제공한다.
  - J2EE 애플리케이션 레퍼런스 아키텍처



❖ 초기 아키텍처 모델 정의

- 레퍼런스 아키텍처 선택
  - J2EE 애플리케이션 레퍼런스 아키텍처에 레이어 아키텍처 스타일(layered architecture style)을 적용할 수 있다.
    - 프리젠테이션 레이어(Presentation Layer)
    - 비즈니스 레이어(Business Layer)
    - 데이터 레이어(Data Layer)
  - 프리젠테이션 레이어(presentation layer)
    - 웹 티어에 위치하며, 사용자가 엔터프라이즈 애플리케이션과 상호작용하기 위해 호출되는 컴포넌트를 포함한다.
  - 비즈니스 레이어(business layer)
    - 비즈니스 로직(business logic)을 구현하며, 비즈니스 컴포넌트(business component)와 비즈니스 워크플로우(business workflow), 서비스 인터페이스(service interface)를 포함한다.
    - 비즈니스 컴포넌트
      ≫ 비즈니스 규칙(business rule)이라고도 하는 비즈니스 로직을 캡슐화하고 구현하는 비즈니스 개념의 소프트웨어 실현(software realization of business concept)이다.
    - 비즈니스 워크플로우
      ≫ 비즈니스가 수행하는 행위 과정 즉, 비즈니스 프로세스를 캡슐화하는 컴포넌트이다.
    - 서비스 인터페이스
      ≫ 외부에 대해 서비스를 제공하는 컴포넌트로서, 일반적으로 서비스 인터페이스는 비즈니스 퍼사드(business façade)로서 역할을 하는 비즈니스 퍼사드 컴포넌트로 실현되며, XML 웹 서비스를 사용하여 실현할 수도 있다.

❖ 초기 아키텍처 모델 정의

- 레퍼런스 아키텍처 선택
  - 데이터 레이어(data layer)
    - 데이터베이스에 저장된 데이터를 비즈니스 레이어 측에 노출하는 데이터 액세스 컴포넌트(data access component)를 포함하며, 서비스 에이전트 컴포넌트(service agent component)를 포함할 수 있다.
    - 데이터 액세스 컴포넌트는 특정한 데이터 저장 솔루션의 세부 사항으로부터 비즈니스 레이어를 분리시키며, 이러한 분리는 다음과 같은 이점을 제공한다.

      ≫ 데이터베이스가 변경될 때 영향을 최소화할 수 있다.

      ≫ 데이터베이스 스키마와 같은 데이터 표현 방식이 변경될 때 영향을 최소화할 수 있다.

      ≫ 한 곳에서 특정한 데이터를 조작하는 코드는 캡슐화함으로써 테스트와 관리를 쉽게 할 수 있다.

    - 서비스 에이전트

      ≫ 서비스 에이전트는 외부 시스템과 상호작용을 전담함으로써 외부 시스템의 변경에 대하여 시스템의 다른 부분을 분리시키는 역할을 하는 컴포넌트이다.

초기 아키텍처 개요 정의

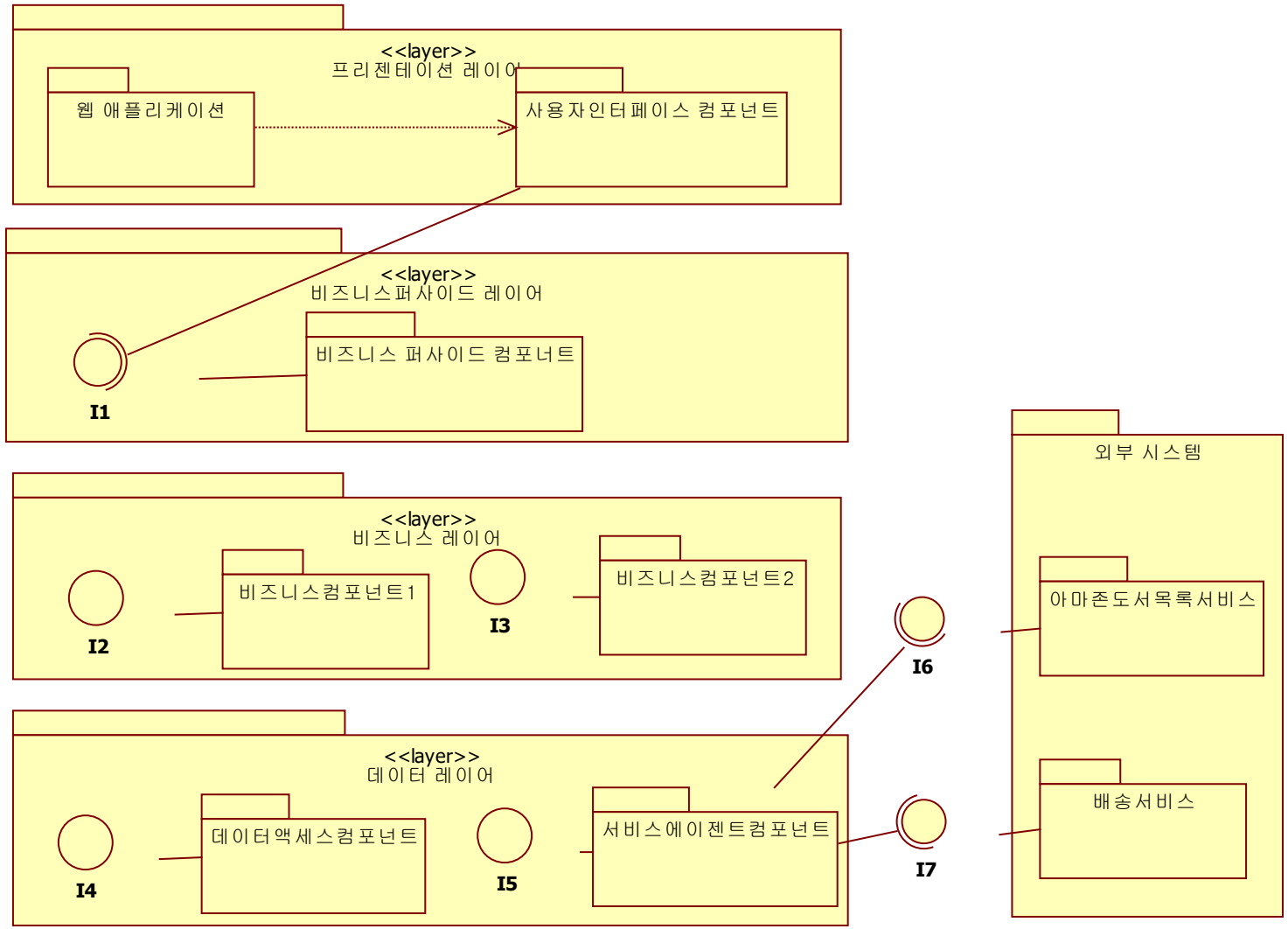
❖ 초기 아키텍처 모델 정의

- 애플리케이션 아키텍처의 논리 뷰(logical view)

- 기술 아키텍처의 배포 뷰(deployment view)

초기 아키텍처 개요 정의

❖ 아키텍처 모델의 논리뷰



아키텍처 모델의 논리뷰

❖ 프리젠테이션 레이어(Presentation Layer)

- 애플리케이션의 사용자 인터페이스를 제공
- 웹 페이지와 같은 사용자 인터페이스를 구성하고 처리하는 사용자 인터페이스 컴포넌트가 포함

❖ 비즈니스 퍼사드 레이어(Business Fasade Layer)

- 비즈니스 레이어에 대한 접근을 클라이언트로부터 분리함으로써 비즈니스 컴포넌트의 변화 시에도 시스템이 유연성을 갖는 것을 도와줌
- 비즈니스 퍼사드 컴포넌트나 웹 서비스 컴포넌트가 포함
  - 비즈니스 퍼사드 컴포넌트는 사용자 인터페이스 컴포넌트로부터의 서비스 요청을 비즈니스 컴포넌트에에 전달하는 역할만 제공
- 비즈니스 퍼사드 컴포넌트에는 비즈니스 로직을 구현하지 않음

❖ 비즈니스 레이어(Business Layer)

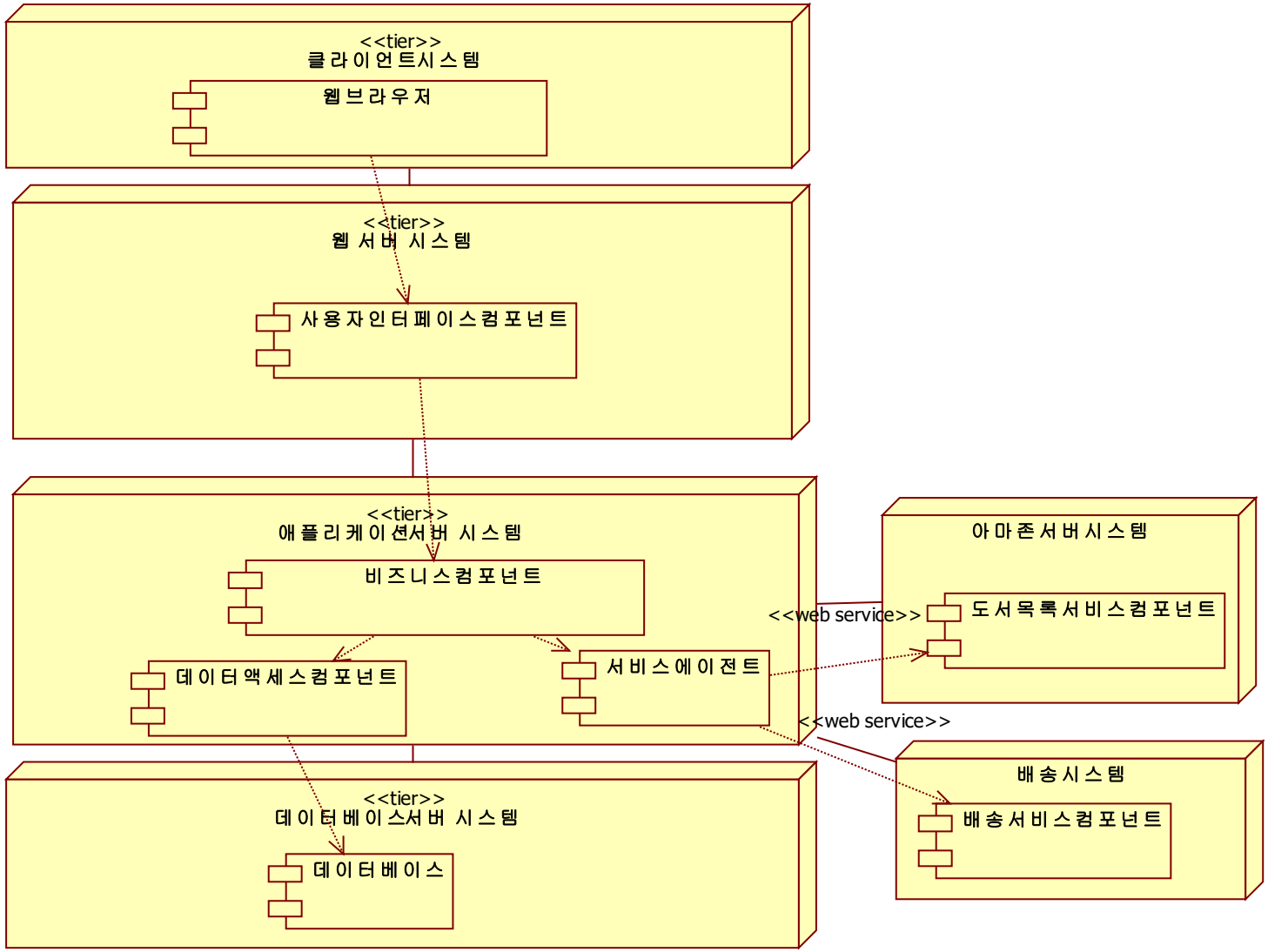
- 애플리케이션의 비즈니스 기능을 구현
- 비즈니스 로직과 프로세스를 구현하는 비즈니스 컴포넌트가 포함

❖ 데이터 레이어(Data Layer)

- 데이터 레이어는 데이터베이스나 웹 서비스 등을 통해 외부 시스템에 접근하는 기능을 제공
- 데이터 저장소에 접근하여 데이터의 입출력 처리를 담당하는 데이터 액세스 컴포넌트(Data Access Component)가 포함되며, 서비스 에이전트 컴포넌트(Service Agent Component)가 아마존 도서 목록 서비스와 배송 서비스 등 외부 시스템과 상호 작용을 전담하게 함으로써 이들 외부 시스템의 변화에도 최대한 유연성을 갖도록 한다.

초기 아키텍처 개요 정의

❖ 아키텍처 모델 의 배포뷰



초기 아키텍처 개요 정의

❖ 초기 아키텍처 모델 정의

- 클라이언트 티어(Client Tier, 클라이언트 시스템)
  - 사용자는 웹 브라우저를 사용하여 웹을 통하여 온라인 서점 관리 시스템에 접근한다

- 웹 애플리케이션 티어(Web Application Tier, 웹 서버 시스템)
  - 웹 애플리케이션을 구성하는 사용자 인터페이스 컴포넌트가 웹 서버 상에 배포된다

- 애플리케이션 티어(Application Tier, 애플리케이션 서버)
  - 비즈니스 컴포넌트가 웹 애플리케이션 서버 상에서 배포된다.
  - 비즈니스 컴포넌트는 다음과 같이 목적에 따라 4개의 레이어로 분류된다.
  - 비즈니스 퍼사드 레이어(Business Façade Layer)
    - 비즈니스 퍼사드 레이어는 비즈니스 컴포넌트 이하 레이어에 대한 접근을 클라이언트로부터 분리함으로써 비즈니스 컴포넌트의 변화 시에도 시스템이 유연성을 갖는 것을 도와준다.
  - 비즈니스 레이어(Business Layer)
    - 비즈니스 레이어의 비즈니스 컴포넌트는 비즈니스 로직과 프로세스를 처리한다.
  - 데이터 레이어(Data Layer)
    - 데이터 액세스 컴포넌트가 데이터 저장소에 접근하여 데이터의 입출력 처리를 담당하며,
    - 서비스 에이전트 컴포넌트 레이어는 아마존 도서 목록 서비스와 배송 서비스 등 외부 시스템과 상호작용을 전담함으로써 이들 외부 시스템의 변화에도 최대한 유연성을 갖도록 한다.
- 데이터 티어(Data Tier, 데이터베이스 서버)
  - 데이터베이스가 데이터베이스서버 상에서 배포된다.

❖ 초기 아키텍처 모델 정의

- 아키텍처명세서 작성
  - 초기 아키텍처 모델이 정의되면 소프트웨어아키텍처명세서에 초기 아키텍처 모델을 기술한다.
  - 초기 아키텍처 모델 논리 뷰
    - 애플리케이션 아키텍처의 초기 아키텍처 모델에 기술한다.
  - 초기 아키텍처 모델 배포 뷰
    - 기술 아키텍처의 초기 배포 모델에 기술한다.
  - 이 시점에서 소프트웨어아키텍처명세서가 처음 생성되므로, 개요와 비즈니스 아키텍처를 소프트웨어아키텍처명세서에 기술한다.
  - 개요
    - 목적과 범위, 아키텍처에 영향을 미치는 주요 사용자 요구사항 및 제약사항 그리고 품질 속성이 기술된다.
  - 비즈니스 아키텍처
    - 요구 파악 단계에서 정의된 유스케이스 모델이 기술된다.

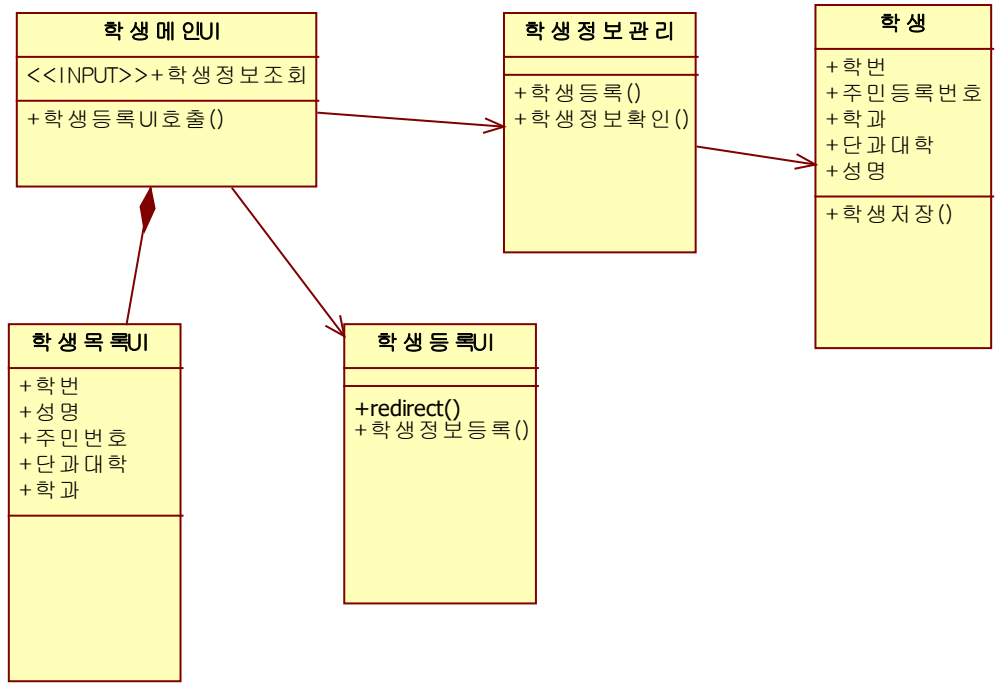
❖ 유스케이스 분석

- 작업의 목표
  - 유스케이스 실현을 통해 유스케이스의 행위를 설명
  - 생성된 유스케이스 실현은 비즈니스 컴포넌트의 후보를 도출하는데 사용되며, 비즈니스 객체 모델을 생성하는 기반

- 유스케이스 분석은 다음과 같은 절차로 진행한다.
  ❶ 유스케이스 실현을 생성
  ❷ 유스케이스 행위에서 분석 클래스를 식별
  ❸ 분석 클래스 사이의 상호작용을 기술함으로써 유스케이스를 실현
  ❹ 분석 클래스 모델을 생성

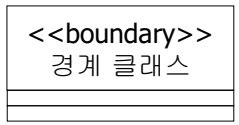
❖ 유스케이스 분석

- 유스케이스 실현 생성
  - 유스케이스 분석(use case analysis) 작업은 요구 파악 단계에서 도출된 각 유스케이스에 대하여 유스케이스 실현(use case realization)을 생성함으로써 시작한다.
  - 유스케이스 실현



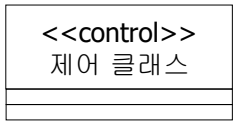
❖ 유스케이스 분석

- 분석 클래스 식별
  - MVC 패턴을 적용하여 세 가지 유형의 클래스를 식별한다.
    - 경계 클래스(boundary class)
    - 제어 클래스(control class)
    - 실체 클래스(entity class)
- 분석 클래스
  - 유스케이스가 실행하는 행위에 대해서만 관심을 갖는다.
  - RUP의 분석 클래스
    - 행위뿐만 아니라, 애트리뷰트에도 관심을 갖는다.
  - 경계 클래스
    - 사용자 인터페이스 클래스(user interface class)
      - ≫ 사용자가 시스템에서 사용하게 될 화면
      - ≫ 경계 클래스를 찾는 일에만 집중한다.
    - 시스템 인터페이스 클래스(system interface class)
      - ≫ 기존의 외부 시스템과 커뮤니케이션하는 경계 클래스
      - ≫ 유스케이스 모델에서는 액터(actor)로 정의되어 있는 경우가 많다.
      - ≫ 기존 시스템에 인터페이스가 정의되어 있지 않다면 시스템 인터페이스 클래스가 기존 시스템에 대한 어댑터(adapter)로서의 역할을 할 수 있도록 인터페이스 행위를 도출해야 한다.



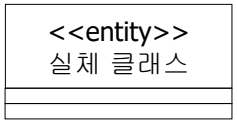
❖ 유스케이스 분석

- 분석 클래스 식별
  - 제어 클래스
    - 시스템을 제어하는 행위를 제공한다.
    - 비즈니스 컴포넌트의 후보가 된다.
    - 유스케이스 행위에서 제어 클래스를 찾을 때 다음 사항을 고려한다.
      - ≫ 각 유스케이스에 대하여 하나의 제어 클래스를 생성한다.
      - ≫ CRUD(Create/Read/Update/Delete) 유스케이스라면 제어 클래스가 반드시 필요한 것은 아니다.
      - ≫ 그러나 제어 클래스는 비즈니스 컴포넌트의 유력한 후보가 되므로, 일단 CRUD 유스케이스에 대해서도 제어 클래스를 도출하고 나중에 제외시킬 것을 고려하는 것이 좋다.
      - ≫ 복잡한 유스케이스라면 하나 이상의 제어 클래스가 필요할 수도 있다.
    - 일반적으로 제어 클래스는 다음과 같은 특징을 갖는다.
      - ≫ 환경 독립적인 행위를 제공한다.
      - ≫ 유스케이스 내부의 제어 로직(이벤트 사이의 순서)과 트랜잭션을 정의한다.
      - ≫ 실체 클래스의 내부 구조 또는 행위가 변경되어도 그다지 변경되지 않는다.
      - ≫ 여러 실체 클래스를 사용함으로써 유스케이스의 작업 흐름을 제어하는 행위를 제공한다.



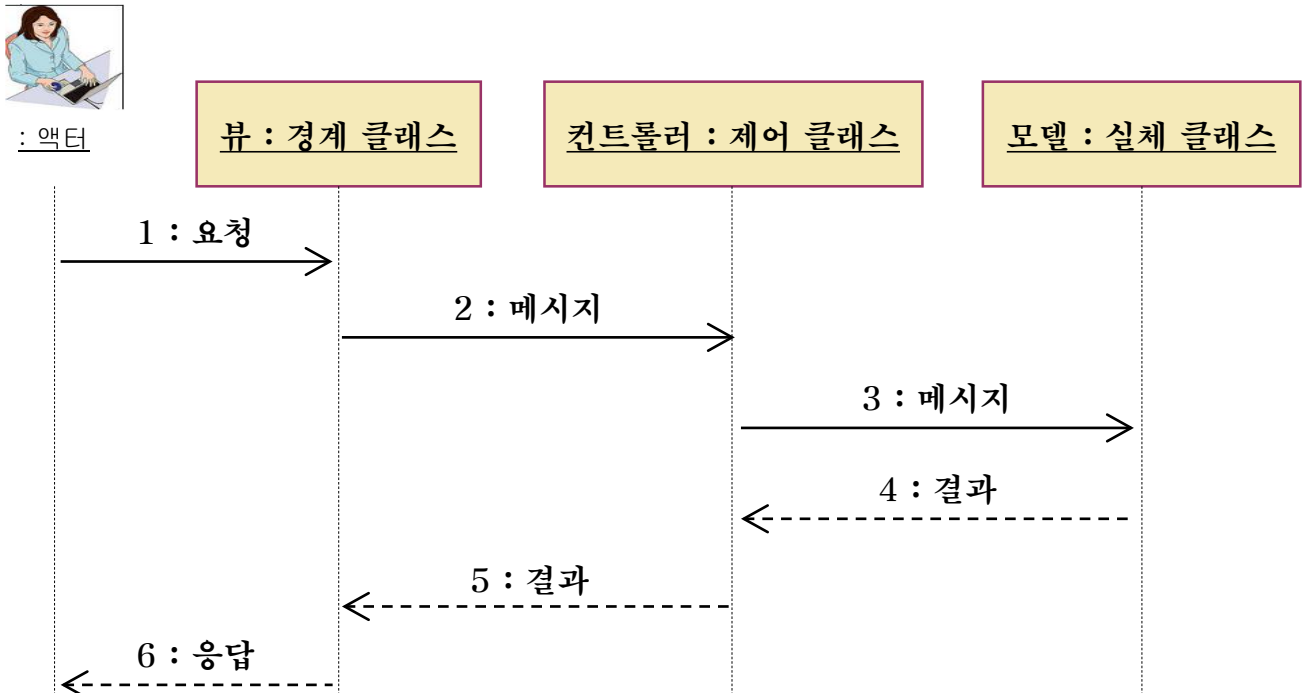
❖ 유스케이스 분석

- 분석 클래스 식별
  - 실체 클래스
    - 시스템에 저장되는 정보를 표현한다.
    - 일반적으로 시스템이 관리하는 주요 추상화 개념을 표현하기 위해 사용된다.
    - 실체 클래스는 비즈니스 객체 모델 생성 작업에서 도출될 비즈니스 객체의 유력한 후보가 되며, 데이터 액세스 컴포넌트(data access component)의 후보가 된다.
    - 실체 클래스는 보통 수동적인 지속성을 가지며, 시스템에 정보를 저장하고 관리하는 역할을 한다.



❖ 유스케이스 분석

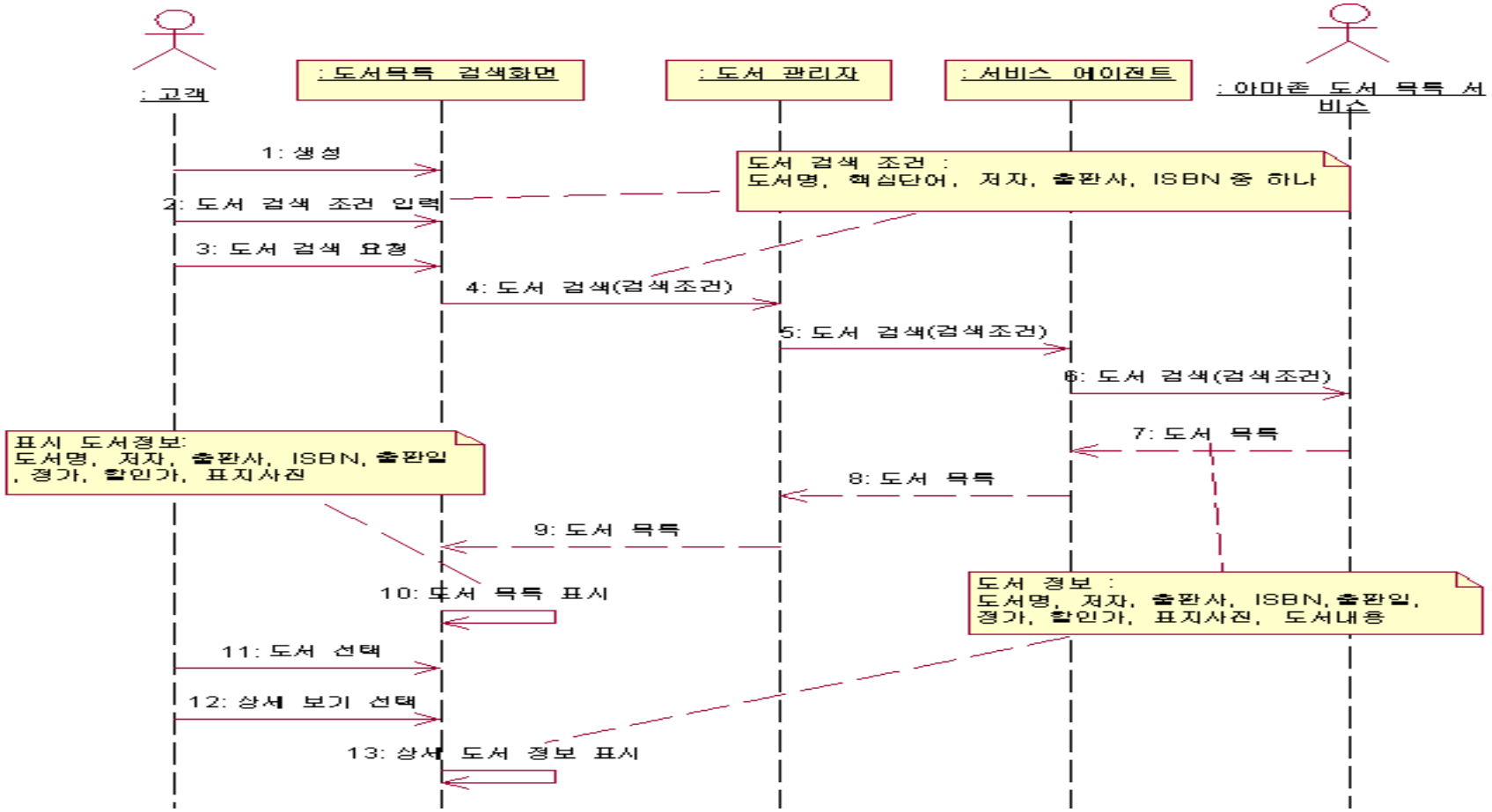
- 유스케이스 실현
  - 각 유스케이스에 대하여 하나 이상의 시퀀스 다이어그램을 생성해야 한다.
  - 시퀀스 다이어그램



- 유스케이스의 주 이벤트 흐름에 대하여 하나의 시퀀스 다이어그램이 필요하고, 여기에 각 대체 또는 예외 흐름에 대하여 각각 시퀀스 다이어그램을 생성할 필요가 있다.
- 또한, 복잡하여 하나의 다이어그램으로는 쉽게 파악할 수 없는 유스케이스 이벤트 흐름에 대해서도 이벤트 흐름을 간단하게 하기 위해 별도의 시퀀스 다이어그램을 생성한다.

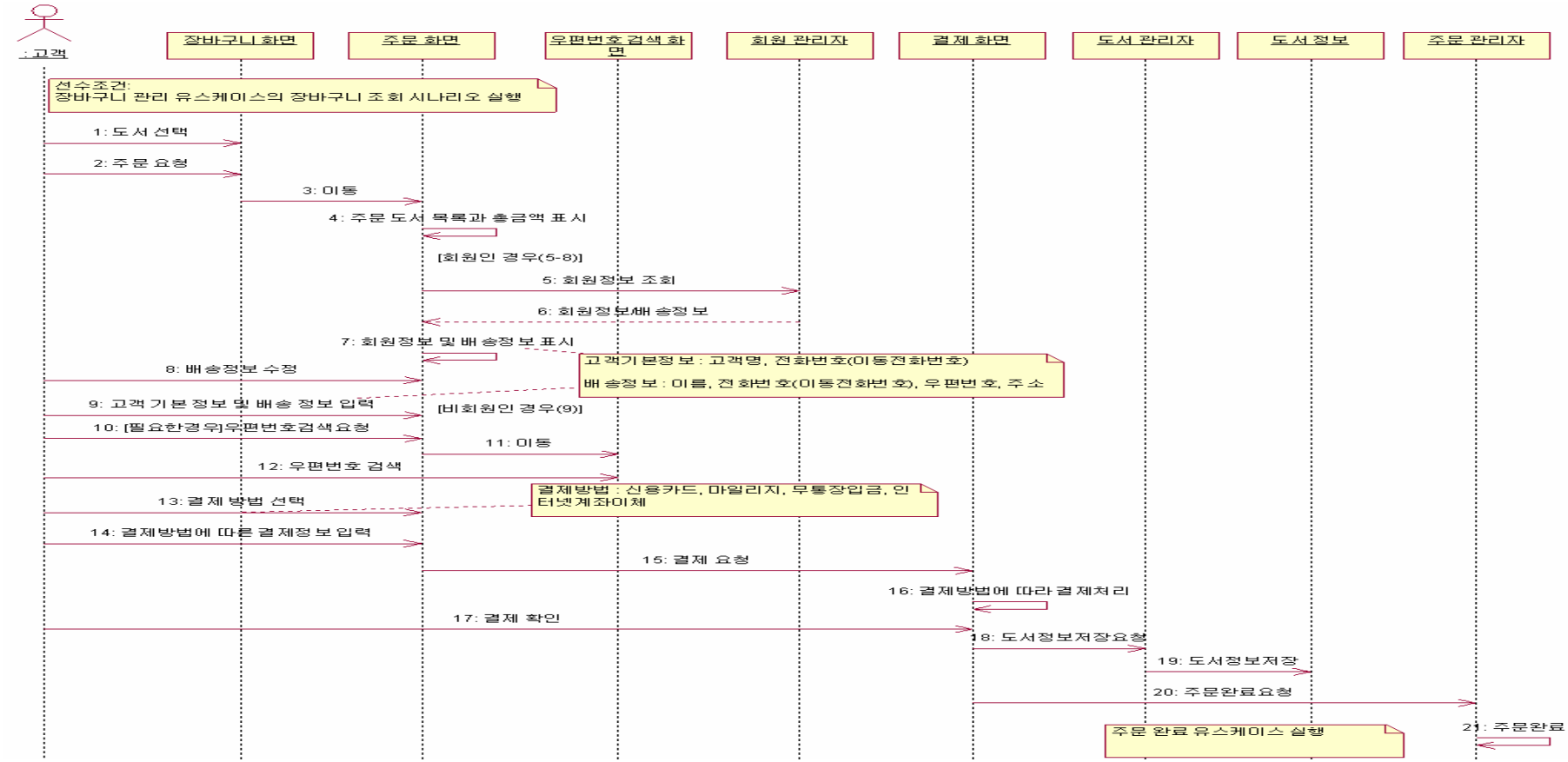
❖ 유스케이스 분석

- 유스케이스 실현 - 도서 목록 검색 유스케이스 실현



❖ 유스케이스 분석

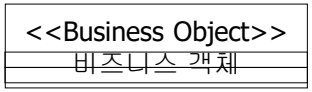
- 유스케이스 실현
  - 도서 주문 유스케이스 실현



❖ 비즈니스 객체 모델 생성

- 업무 처리에서 중요한 개념을 갖는 비즈니스 객체(business object)를 도출하고, 이들 비즈니스 객체 사이의 관계를 정의한다.
- 이 작업에서 생성된 비즈니스 객체 모델은 데이터베이스 설계 활동의 중요한 입력 산출물이 되며, 비즈니스 컴포넌트를 도출하기 위한 입력으로도 활용된다.
- 비즈니스 객체를 도출하는 방법
  - 유스케이스 분석 작업에 도출된 전체 유스케이스 실현을 대상으로 제어 분석 클래스나 실체 분석 클래스에 메시지와 함께 전달되는 데이터를 그룹화하여 비즈니스 객체를 정의한다.
  - 유스케이스 실현에서의 메시지 흐름을 참조하여 이들 비즈니스 객체 사이의 관계를 도출하고 정의한다.

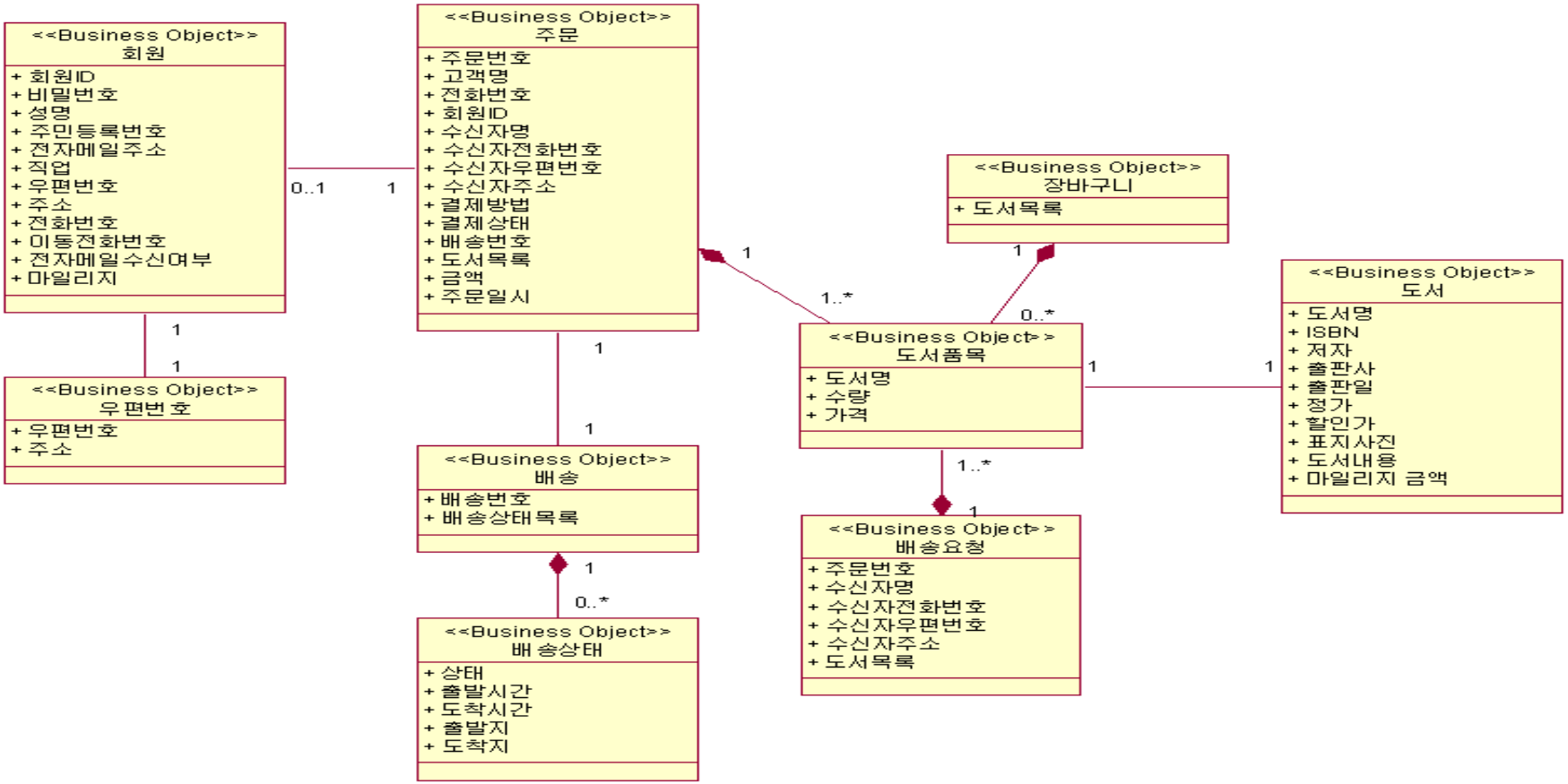
- 비즈니스 객체는 다음과 같이 <<Business Object>> 스테레오타입을 갖는 클래스로 표현한다.



행위 분석

❖ 비즈니스 객체 모델 생성

- 비즈니스 객체 모델



❖ 비즈니스 객체 모델 생성

– 비즈니스 객체 모델

회 원	회원의 이름과 주민등록번호, 전자메일주소, 직업, 우편번호, 주소, 전화번호, 이동전화번호, 전자메일을 수신할 지의 여부, 확보한 마일리지 등의 정보를 표현한다.
도 서	고객이 검색 또는 주문한 도서의 정보를 표현한다. 여기에는 도서명, ISBN, 저자, 출판사, 출판일, 정가, 할인가, 표지사진, 도서내용, 구입 시 고객이 받게 되는 마일리지 금액 등의 정보가 포함된다.
주 문	주문 고객명과 전화번호 및 수신자의 이름과 전화번호, 우편번호, 주소가 포함되며, 결제 방법, 결제 완료 여부를 나타내는 상태, 배송 확인을 위한 배송번호 등의 주문 관련 정보도 포함된다. 또한, 고객이 주문한 도서 목록과 이들 도서의 총합계 금액, 주문일시 정보도 포함된다.
배송요청	고객이 배송 요청을 한 정보를 표현한다. 여기에는 주문번호와 함께, 이름과 전화번호, 우편번호, 주소, 그리고 고객이 주문한 도서 목록이 포함된다.
장바구니	고객이 선택하여 장바구니에 담은 도서의 목록을 관리한다.
도서품목	고객이 주문 또는 배송 요청을 한 도서 품목에 대한 정보를 포함한다. 여기에는 도서명, 수량, 가격 등이 포함된다. 주문 비즈니스 객체와 배송요청 비즈니스 객체는 최소한 하나 이상의 도소 품목 비즈니스 객체를 포함한다. 장바구니 비즈니스 객체는 도서 품목 비즈니스 객체를 포함하지 않을 수도 있다.
배 송	고객이 주문한 도서의 주문에서부터 현재까지의 배송 현황을 각 단계별로 표현하는 배송 상태 목록을 관리한다.
배송상태	고객이 주문한 도서의 각 단계 배송 상태를 표현한다.
우편번호	우편번호와 주소 정보를 표현한다.

애플리케이션 아키텍처 설계

❖ 목적

- 비즈니스 컴포넌트를 도출
- 비즈니스 컴포넌트 모델을 생성
- 논리적인 관점에서 비즈니스 컴포넌트를 설계

❖ 작업

- 후보 비즈니스 컴포넌트 도출
- 비즈니스 컴포넌트 모델 정의
- 비즈니스 컴포넌트 설계

❖ 입력 산출물

- 유스케이스기술서
- 초기 애플리케이션 아키텍처 정의서

❖ 출력 산출물

- 비즈니스컴포넌트설계서
- 애플리케이션 아키텍처 정의서

❖ 비즈니스 컴포넌트

- 비즈니스 컴포넌트 정의
  - 비즈니스 개념 또는 비즈니스 프로세스를 구현하는 소프트웨어 구현으로써, 재사용할 수 있는 엔터프라이즈 시스템의 요소로써 해당 비즈니스 개념을 표현하고 구현 및 배포하는데 필요한 모든 소프트웨어 산출물들로 구성됨

- 이와 같은 비즈니스 컴포넌트에 대한 정의는 비즈니스 컴포넌트가 물리적인 개념 뿐만 아니라, 논리적인 개념도 함께 포함하고 있다는 것을 말해준다.
- 비즈니스 컴포넌트를 논리적인 컴포넌트(logical component)로써 하나의 비즈니스 개념을 실현하는 사양(specification)으로 이해한다.
- 개념적인 비즈니스 객체는 논리적인 비즈니스 컴포넌트로 실현되며, 다시 물리적인 구현 컴포넌트로 구현

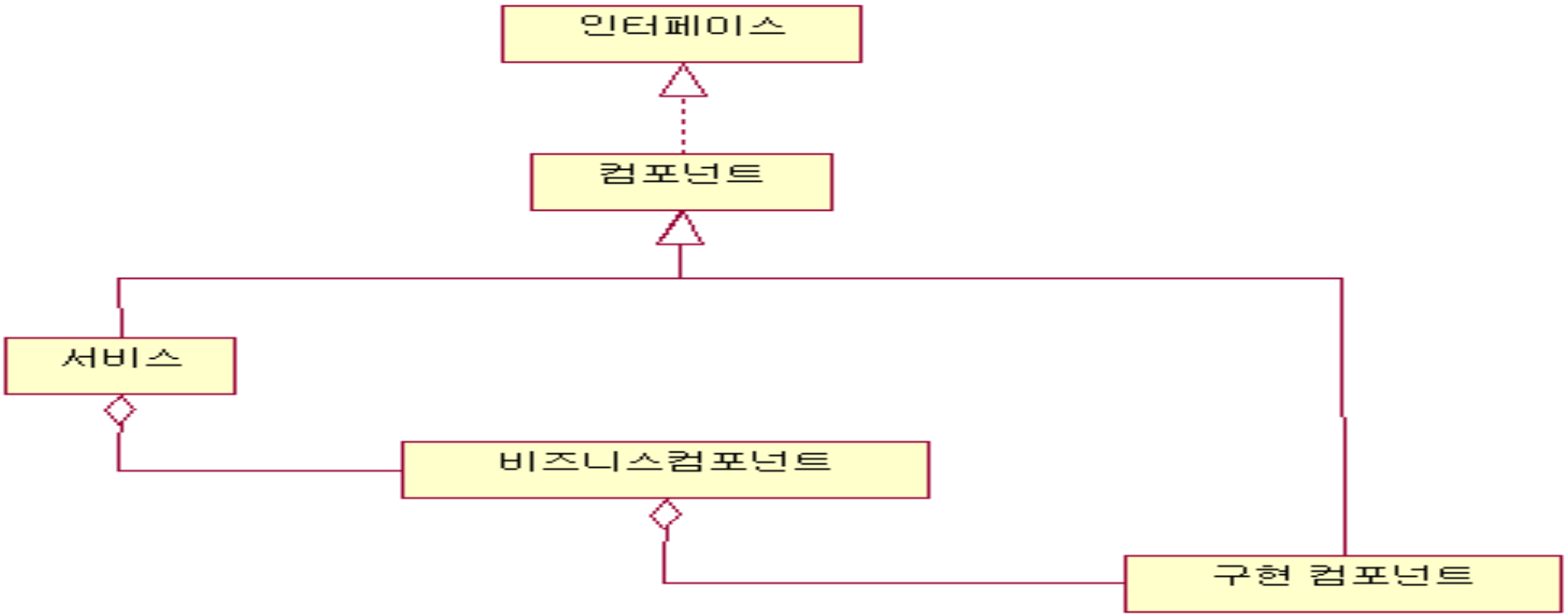
❖ 비즈니스 컴포넌트

- 비즈니스 컴포넌트 모델과 구현 컴포넌트 모델
  - 논리적인 비즈니스 컴포넌트와 이들 사이의 관계는 비즈니스 컴포넌트 모델(business component model)에 정의되며, 물리적인 구현 컴포넌트와 이들 사이의 관계는 구현 컴포넌트 모델(implementation component model)에 정의
  - MDA(Model Driven Architecture)의 PIM(Platform Independent Model)과 PSM(Platform Specific Model)의 개념과 유사
  - MDA의 PIM은 구현 기술에 독립적인 상위 레벨의 추상화 모델
  - 이러한 PIM은 하나 이상의 PSM으로 변형되어 하나의 특정한 구현 기술에서 사용할 수 있는 구현 구조로 시스템을 정의

개발방법론	비즈니스 컴포넌트 모델	구현 컴포넌트 모델
애플리케이션 아키텍처	논리 뷰(logical view) 애플리케이션 모델	물리 뷰(physical view) 구현 모델
MDA	PIM (Platform Independent Model)	PSM (Platform Specific Model)

❖ 비즈니스 컴포넌트

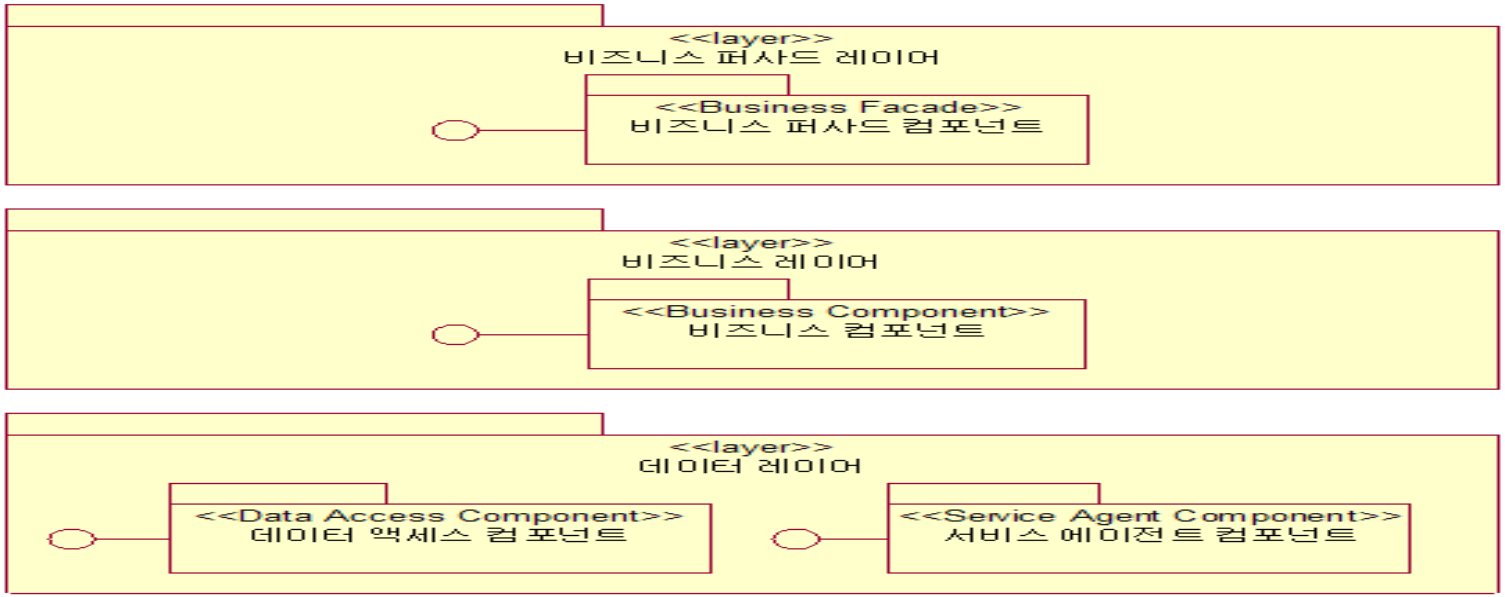
- 비즈니스 컴포넌트와 구현 컴포넌트 그리고 서비스
  - 비즈니스 컴포넌트는 하나 이상의 구현 컴포넌트로 구성되어 하나의 비즈니스 개념을 실현
  - 서비스는 하나 이상의 비즈니스 컴포넌트로 구성되며, 웹 서비스 등의 기술을 사용하여 다른 엔터프라이즈 시스템에게 서비스를 제공



❖ 비즈니스 컴포넌트

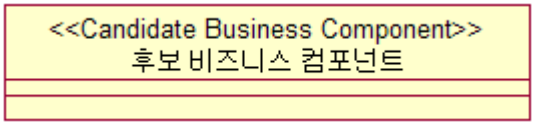
- 비즈니스 컴포넌트

레이어	컴포넌트	스테레오타입
비즈니스 퍼사드 레이어	비즈니스 퍼사드 컴포넌트	<<Business Façade>>
비즈니스 레이어	비즈니스 컴포넌트	<<Business Component>>
데이터 레이어	데이터 액세스 컴포넌트	<<Data Access Component>>
	서비스 에이전트 컴포넌트	<<Service Agent Component>>



❖ 후보 비즈니스 컴포넌트 도출

- 이 작업에서 도출된 후보 비즈니스 컴포넌트는 다시 정제되어 비즈니스 컴포넌트로 선택되어 정의
- 후보 비즈니스 컴포넌트



- 후보 비즈니스 컴포넌트 도출
  - 대부분의 경우에 있어서, 유스케이스 분석 작업에서 도출된 분석 클래스 중에서 제어(control) 클래스가 비즈니스 컴포넌트의 최우선 후보가 된다. CRUD 유스케이스라면 실체(entity) 클래스가 비즈니스 컴포넌트의 후보가 됨

❶ 서로 관련된 유스케이스들의 유스케이스 실현에서 유사한 메시지 흐름을 갖는 제어 클래스를 그룹화하여 하나의 후보 비즈니스 컴포넌트로 정의
  - 유사한 책임(responsibility)은 같은 비즈니스 컴포넌트에 속해야 한다.

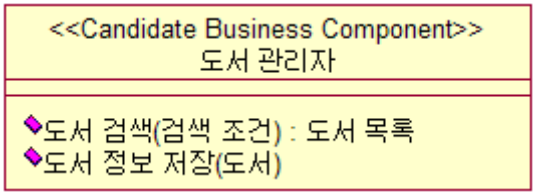
❷ 유사한 메시지 흐름을 갖는 실체 클래스를 그룹화
  - 비즈니스 컴포넌트가 여러 실체를 관리해야 한다면, 이들 실체를 관리하는데 필요한 책임은 해당 비즈니스 컴포넌트에 있음
  - 예
    - ≫ 도서 목록 검색 유스케이스 실현 - 도서 관리자, 서비스 에이전트
    - ≫ 도서 주문 유스케이이스 실현 - 회원 관리자, 도서 정보, 주문 관리자

❖ 후보 비즈니스 컴포넌트 도출

- 후보 비즈니스 컴포넌트 도출

❸ 이들 분석 클래스에게 전달되는 메시지를 분석하여 후보 비즈니스 컴포넌트의 책임(responsibility)으로 분배

- 도서 관리자 후보 비즈니스 컴포넌트



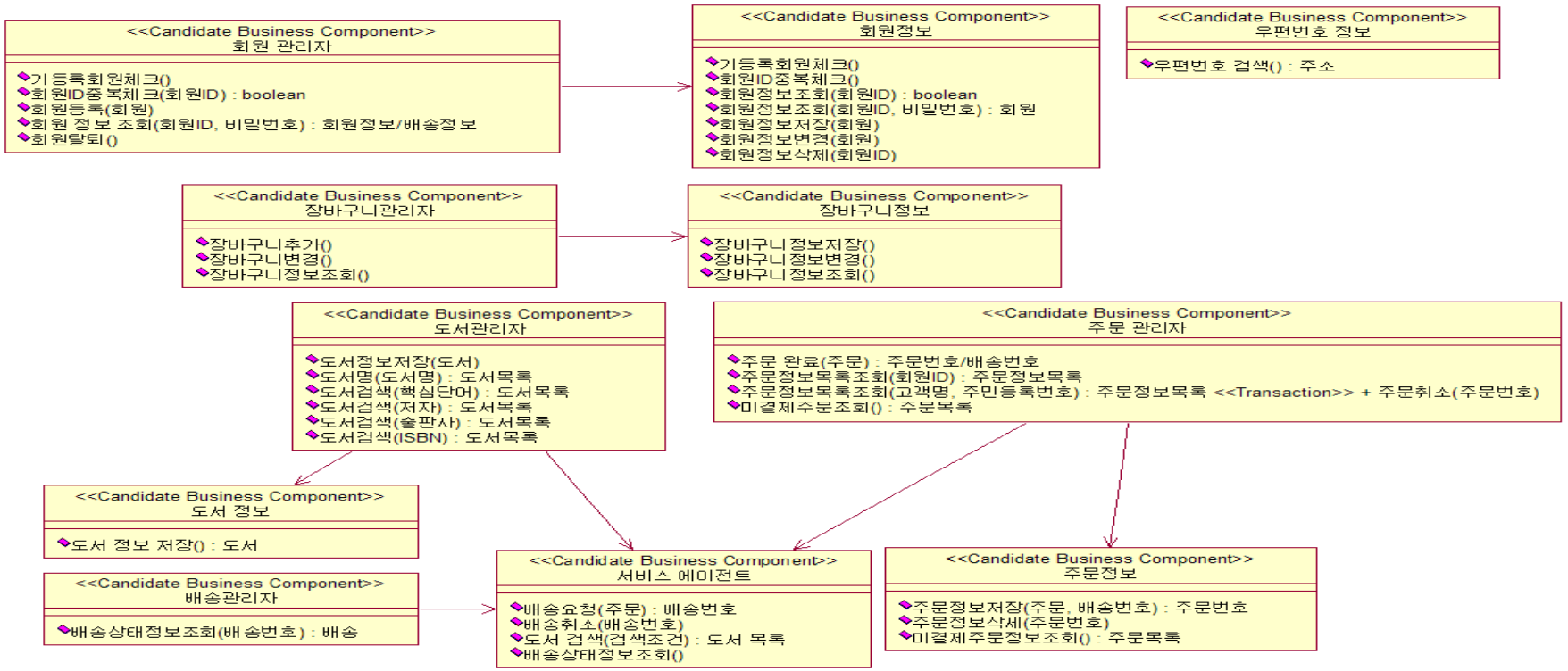
- 후보 비즈니스 컴포넌트 도출 및 책임 할당

❖ 후보 비즈니스 컴포넌트 도출

- 후보 비즈니스 컴포넌트 도출

❹ 후보 비즈니스 컴포넌트 사이의 관계를 찾아서 후보 비즈니스 컴포넌트 모델에 기술

- 유스케이스 실현에서 후보 비즈니스 컴포넌트의 대상이 되는 분석 클래스 사이의 메시지 흐름을 분석함으로써 관계를 찾을 수 있음



❖ 비즈니스 컴포넌트 모델 정의

- 목표
  - 비즈니스 컴포넌트를 도출하고 비즈니스 컴포넌트가 실현할 인터페이스를 정의한 후, 인터페이스를 중심으로 컴포넌트 사이의 관계를 식별하여 비즈니스 컴포넌트 모델을 구조화

- 비즈니스 컴포넌트 모델 정의 작업 절차
  1. 비즈니스 컴포넌트를 도출하고, 비즈니스 인터페이스를 정의
  2. 비즈니스 컴포넌트 사이의 관계를 식별
  3. 비즈니스 컴포넌트 모델을 구조화

- 비즈니스 컴포넌트 도출과 비즈니스 인터페이스 정의 (1)

❶ 후보 비즈니스 컴포넌트 모델에서 아키텍처적으로 중요한 요소를 선택하여 비즈니스 컴포넌트로 도출

- 사용자 요구사항을 충족시킬 수 있는 아키텍처적으로 중요한 요소이어야 한다.
- 예

≫ 후보 비즈니스 컴포넌트 모델에서 '장바구니 관리자'와 '장바구니 정보'는 비즈니스 레이어에서 관리할 필요가 없다고 판단되기 때문에 비즈니스 컴포넌트의 대상에서 제외

≫ 즉, 프리젠테이션 레이어의 요구사항이므로 비즈니스 컴포넌트의 대상이 아님

❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 도출과 비즈니스 인터페이스 정의

❷ 비즈니스 컴포넌트의 대상이 되는 후보 비즈니스 컴포넌트의 책임(responsibility)을 분석하여 비즈니스 컴포넌트가 구현하게 될 비즈니스 인터페이스(business interface)를 정의

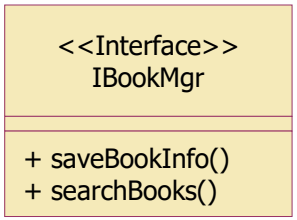
- 후보 비즈니스 컴포넌트에 대하여 대응되는 하나의 비즈니스 인터페이스를 정의
- 이때 중복되는 후보 비즈니스 컴포넌트의 책임은 하나의 비즈니스 인터페이스 행위(behavior)로 통합하여 정의
- 비즈니스 객체 모델 생성 작업에서 도출한 비즈니스 객체 모델도 비즈니스 인터페이스 정의에 반영

❸ 비즈니스 인터페이스명은 대문자 I로 시작하는 영문으로 부여하고, 비즈니스 인터페이스 행위도 후보 비즈니스 컴포넌트의 책임명을 영문으로 바꾸어 이름을 부여

- 예

  ≫ 도서 관리자 후보 비즈니스 컴포넌트에 대한 비즈니스 인터페이스명은 IBookMgr로 부여할 수 있다. Mgr은 Manager 즉, 관리자의 약자로 사용

- 비즈니스 인터페이스 정의



❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 도출과 비즈니스 인터페이스 정의

❹ 후보 비즈니스 컴포넌트에 대하여 비즈니스 인터페이스를 정의

후보 비즈니스 컴포넌트	인터페이스 명
도서 관리자	IBookMgr
회원 관리자	IMemberMgr
주문 관리자	IOrderMgr
배송 관리자	IDeliveryMgr
도서 정보	IBookInfo
회원 정보	IMemberInfo
주문 정보	IOrderInfo
서비스 에이전트	IServiceAgent
우편번호 정보	IPostNoInfo

❖ 비즈니스 컴포넌트 모델 정의

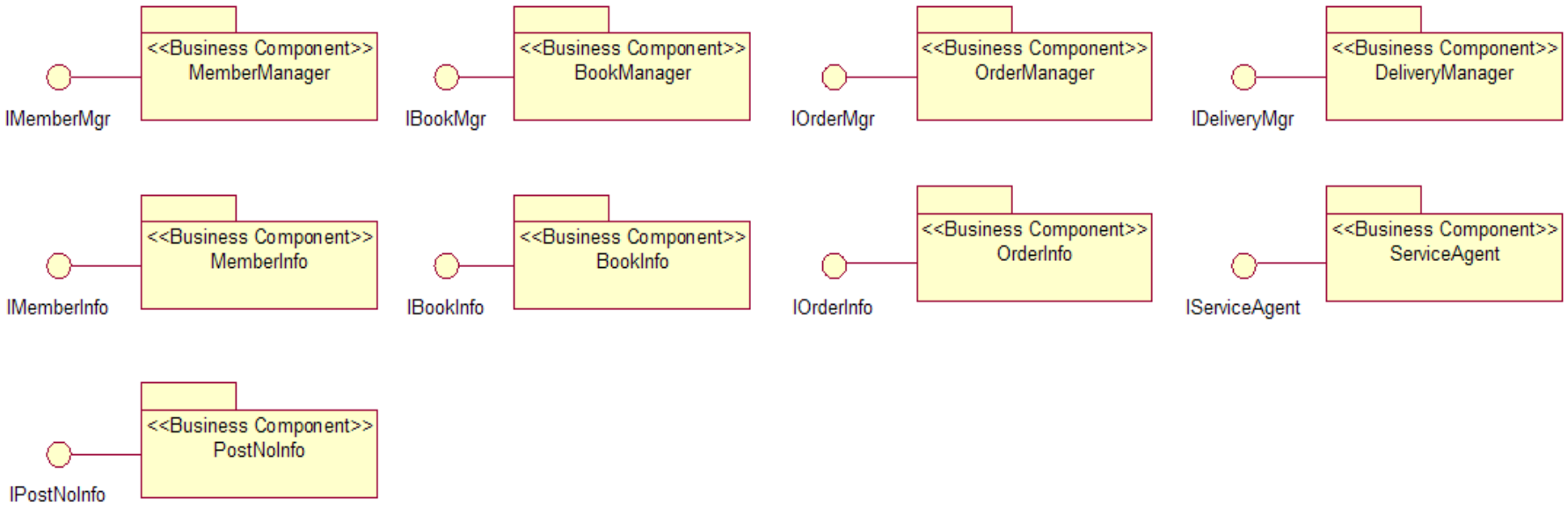
- 비즈니스 컴포넌트 관계 식별

❶ 식별된 인터페이스를 구현하게 될 비즈니스 컴포넌트를 추가하고, 비즈니스 인터페이스와 비즈니스 컴포넌트 사이에 실현(realization) 관계를 정의

한글명	영문명
도서 관리자	BookManager
회원 관리자	MemberManager
주문 관리자	OrderManager
배송 관리자	DeliveryManager
도서 정보	BookInfo
회원 정보	MemberInfo
주문 정보	OrderInfo
서비스 에이전트	ServiceAgent
우편번호 정보	PostNoInfo

❖ 비즈니스 컴포넌트 모델 정의

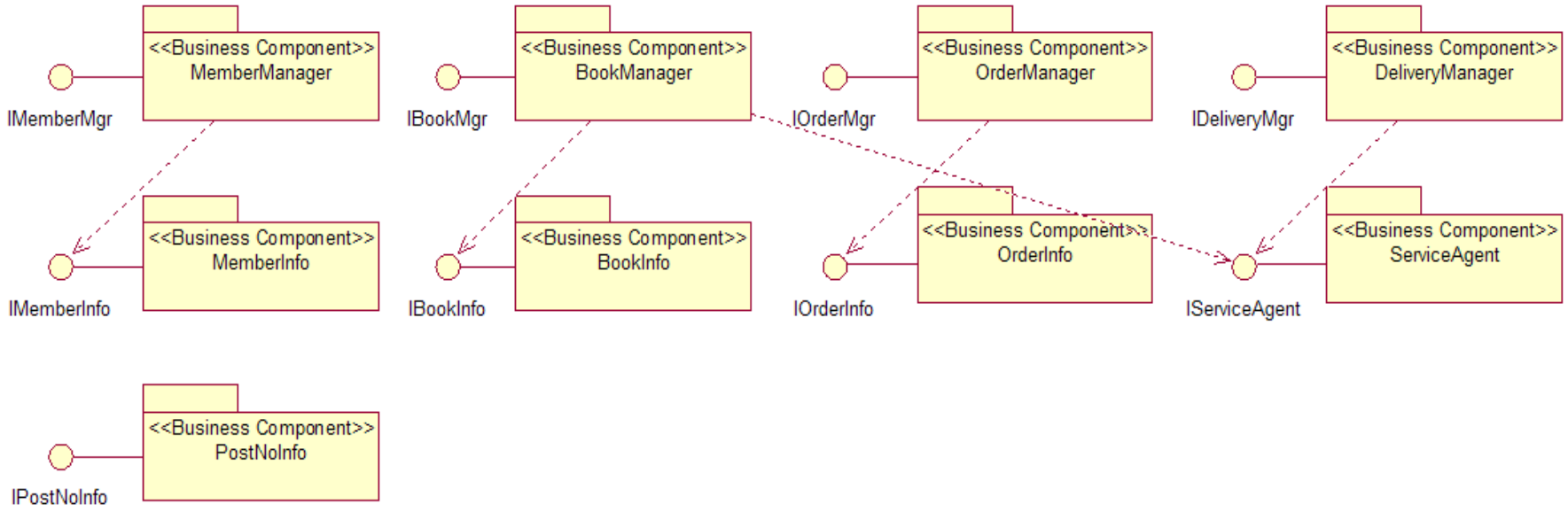
- 비즈니스 컴포넌트 관계 식별



❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 관계 식별

❷ 후보 비즈니스 컴포넌트 모델을 참조하여 비즈니스 인터페이스를 중심으로 비즈니스 컴포넌트 사이의 관계를 식별하고 종속(dependency) 관계를 정의



❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화
  - 유의해야 할 점
    - 비즈니스 컴포넌트 모델이 초기 아키텍처 개요 정의 활동에서 정의한 초기 아키텍처 모델을 충실히 반영해야 한다.
  - 유스케이스 실현 작업에서 MVC(Model-View-Controller) 패턴을 적용하여 경계클래스, 제어클래스, 실체클래스 등으로 분석 클래스를 식별
  - 일반적으로 MVC 패턴과 분석 클래스, 레이어, 컴포넌트 사이에는 다음과 같은 관계

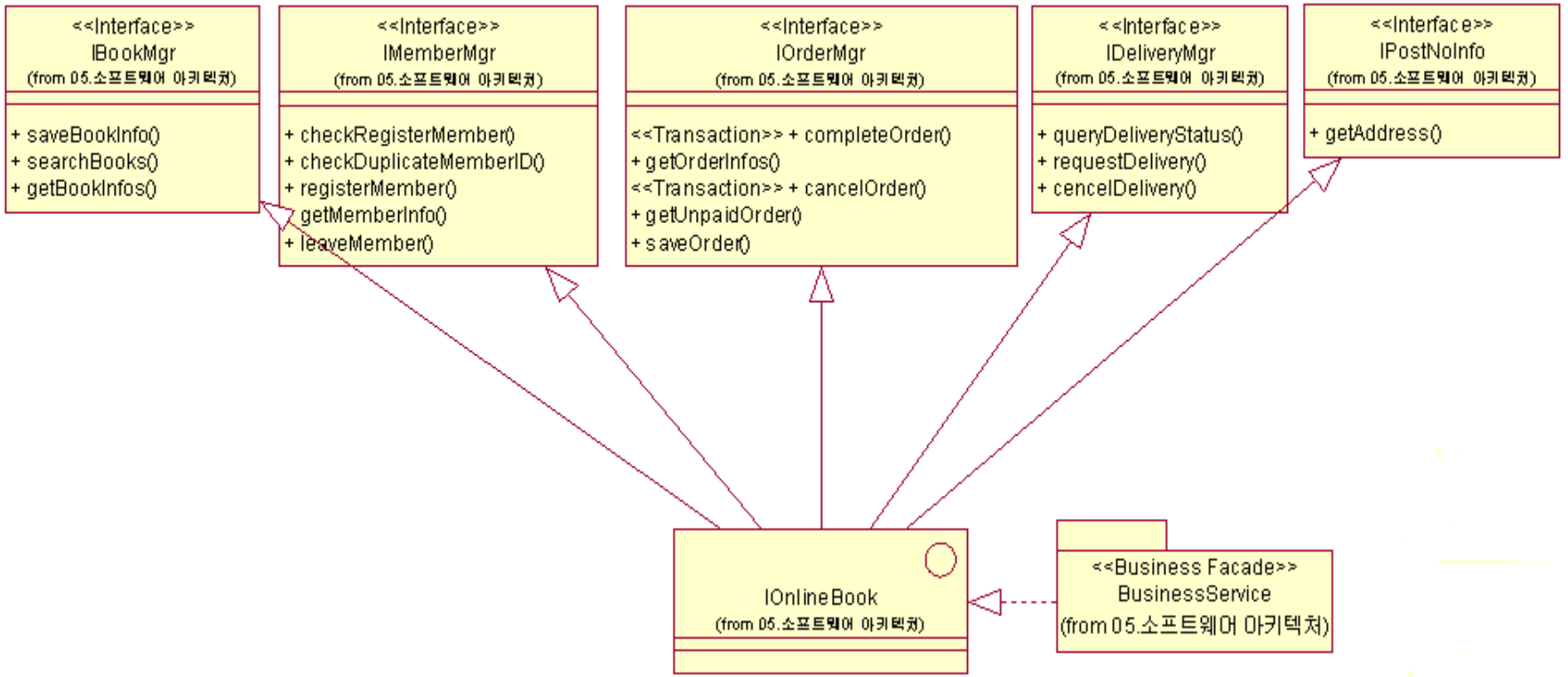
MVC 패턴	분석 클래스	레이어	컴포넌트
뷰(View)	경계 클래스	프리젠테이션 레이어	사용자 인터페이스 컴포넌트
컨트롤러(Controller)	제어 클래스	비즈니스 레이어	비즈니스 컴포넌트
모델(Model)	실체 클래스	데이터 레이어	데이터 액세스 컴포넌트 서비스 에이전트 컴포넌트

  - 이 시점까지 비즈니스 퍼사드 컴포넌트는 비즈니스 로직을 구현하지 않으며, 사용자 인터페이스 컴포넌트로부터의 서비스 요청을 비즈니스 컴포넌트에게 전달시켜주는 역할

  - 비즈니스 퍼사드 컴포넌트의 특징을 반영하여, BusinessService 비즈니스 퍼사드 컴포넌트와 이 비즈니스 퍼사드 컴포넌트가 구현할 IOnlineBook 인터페이스를 정의

❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화
  - 비즈니스 퍼사드 컴포넌트



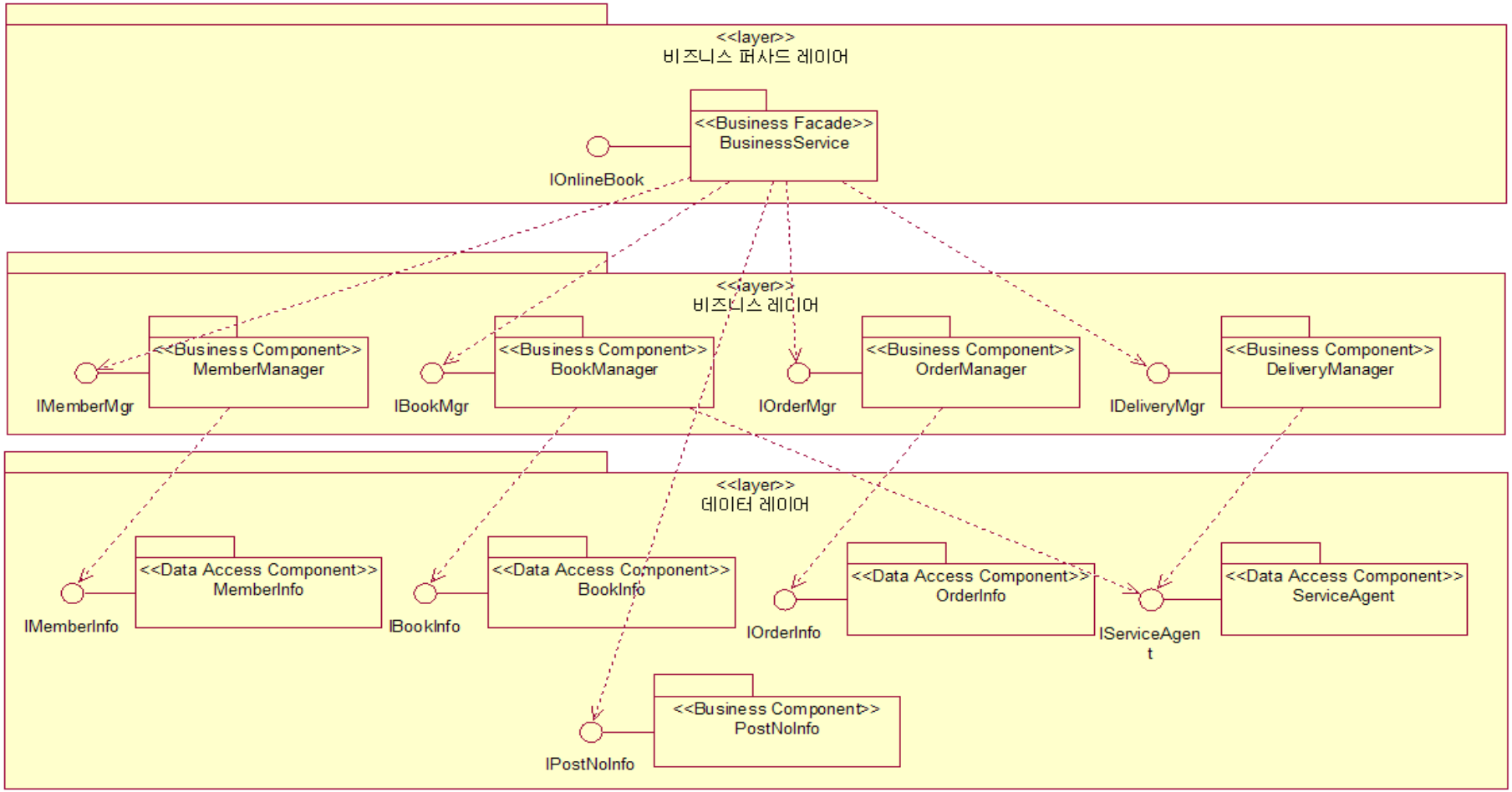
❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화
  - 비즈니스 컴포넌트

레이어	비즈니스 컴포넌트 명		구분 <<스테레오타입>>
	한글	영문	
비즈니스 퍼사드 레이어	비즈니스 서비스	BusinessService	비즈니스 퍼사드 컴포넌트 <<Business Façade Component>>
비즈니스 레이어	회원 관리자	MemberManager	비즈니스 컴포넌트 <<Business Component>>
	도서 관리자	BookManager	
	주문 관리자	OrderManager	
	배송 관리자	DeliveryManager	
데이터 레이어	회원 정보	MemberInfo	데이터 액세스 컴포넌트 <<Data Access Component>>
	도서 정보	BookInfo	
	주문 정보	OrderInfo	
	우편번호 정보	PostNoInfo	
	서비스 에이전트	ServiceAgent	서비스 에이전트 컴포넌트 <<Service Agent Component>>

❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화



❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화
  - 비즈니스 컴포넌트 개요 설명

레이어	비즈니스 컴포넌트 명		컴포넌트 유형	실현 인터페이스
	한글	영문		
비즈니스 퍼사드 레이어	비즈니스 서비스	BusinessService	비즈니스 퍼사드 컴포넌트 <<Business Façade Component>>	IOnlineBook
비즈니스 레이어	회원 관리자	MemberManager	비즈니스 컴포넌트 <<Business Component>>	IMemberMgr
	도서 관리자	BookManager		IBookMgr
	주문 관리자	OrderManager		IOrderMgr
	배송 관리자	DeliveryManager		IDeliveryMgr
데이터 레이어	회원 정보	MemberInfo	데이터 액세스 컴포넌트 <<Data Access Component>>	IMemberInfo
	도서 정보	BookInfo		IBookInfo
	주문 정보	OrderInfo		IOrderInfo
	우편번호 정보	PostNoInfo		IPostNoInfo
	서비스 에이전트	ServiceAgent	서비스 에이전트 컴포넌트 <<Service Agent Component>>	IServiceAgent

❖ 비즈니스 컴포넌트 모델 정의

- 비즈니스 컴포넌트 모델 구조화
  - 비즈니스 컴포넌트 개요 설명

비즈니스 컴포넌트 명	설 명
Business Service(비즈니스 서비스) 컴포넌트	온라인 서점에서 제공하는 비즈니스 컴포넌트에 대한 퍼사드(Façade)로서의 역할을 하는 비즈니스 퍼사드 컴포넌트이다.
MemberManager(회원관리자) 컴포넌트	온라인 서점에 회원으로 등록하고자 하는 고객을 관리하는 비즈니스 컴포넌트이다.
BookManager(도서관리자) 컴포넌트	서비스 에이전트 컴포넌트를 통해 아마존 도서 목록 서비스에서 도서 목록을 검색하고, 주문된 도서 정보를 저장하는 기능을 제공하는 비즈니스 컴포넌트이다.
OrderManager(주문관리자) 컴포넌트	장바구니에서 고객이 선택한 도서를 주문받고, 고객이 주문한 주문 정보를 조회하고, 취소하는 기능을 제공하는 비즈니스 컴포넌트이다. 이와 함께, 고객이 자동으로 입금 확인이 되지 않는 방법으로 결제한 주문에 대하여 주문 완료 처리를 위하여 미결제된 주문 내역을 조회하는 기능도 제공한다.
DeliveryManager(배송관리자) 컴포넌트	서비스 에이전트 컴포넌트를 통해 배송 업체에서 고객이 주문한 도서의 현재 배송 상태를 조회하고, 도서의 배송을 요청 또는 취소하는 기능을 제공하는 비즈니스 컴포넌트이다.
MemberInfo(회원정보) 컴포넌트	회원 정보를 저장, 변경, 삭제, 조회하는 데이터 액세스 컴포넌트이다.
BookInfo(도서정보) 컴포넌트	주문된 도서 정보를 저장하는 데이터 액세스 컴포넌트이다.

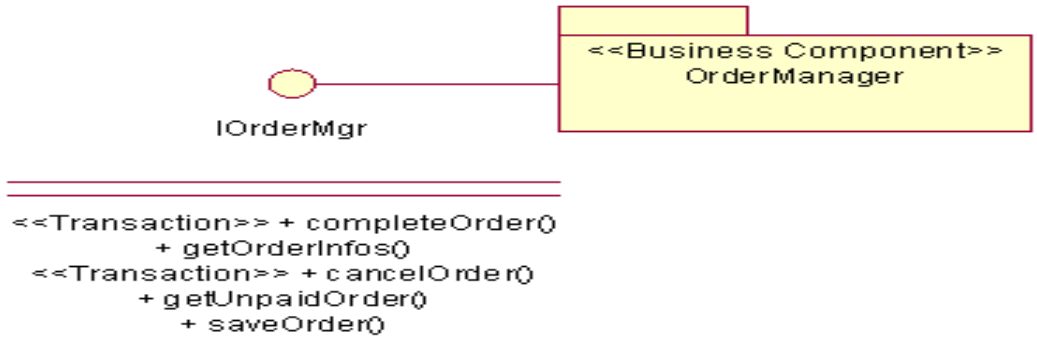
❖ 비즈니스 컴포넌트 설계

- 비즈니스 컴포넌트가 실현할 인터페이스를 명세화하여 비즈니스컴포넌트설계서에 기술
  - 비즈니스컴포넌트설계서를 생성하고, 유스케이스 실현을 참고하여 인터페이스에 포함된 각 행위의 기능을 비즈니스컴포넌트설계서에 상세히 기술
  - 입, 출력 매개변수, 컴포넌트가 상태를 유지해야 할 항목 및 컴포넌트에 적용해야 할 제약사항 등이 기술
- 비즈니스 컴포넌트 설계(주문 관리자 비즈니스 컴포넌트의 예)

◆ 주문 관리자(OrderManager) 컴포넌트

➢ 개요

장바구니에서 고객이 선택한 도서를 주문받고, 고객이 주문한 주문 정보를 조회하고, 취소하는 기능을 제공하는 비즈니스 컴포넌트이다. 이와함께, 고객이 자동으로 입금 확인이 되지 않는 방법으로 결제한 주문에 대하여 주문 완료 처리를 위하여 미결제된 주문 내역을 조회하는 기능도 제공한다.



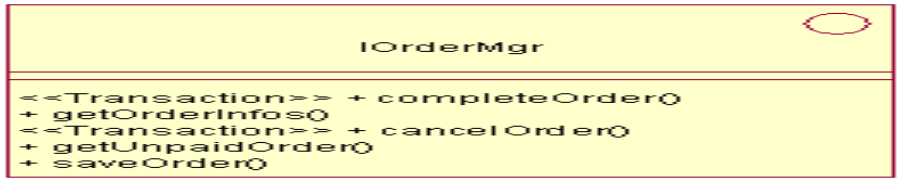
❖ 비즈니스 컴포넌트 설계

- 비즈니스 컴포넌트 설계(주문 관리자 비즈니스 컴포넌트의 예)

➢ ***실현 인터페이스 목록***

인터페이스명	주요 내용
IOrderMgr	주문 정보를 등록, 조회, 취소하고, 미결제된 주문 내역을 조회하는 기능을 제공한다.

➢ *IOrderMgr* ***인터페이스 명세***



➢ *completeOrder* ***오퍼레이션***

설명	주문 처리를 완료한다.
입력값	주문 비즈니스 객체(고객명, 전화번호, 회원ID, 수신자명, 수신자전화번호, 수신자주소, 결제방법, 결제상태, 도서목록, 금액, 주문일시)
출력값	없음
상태유지	없음
제약사항	책임1에 책임3까지는 하나의 트랜잭션 안에서 처리되어야 한다.
책임1	배송 관리자 컴포넌트를 통해 배송 업체에 배송 요청을 한다 .
책임2	비회원이 주문한 경우 비회원으로 회원 정보에 저장한다.
책임3	주문 비즈니스 객체 정보를 저장한다.

애플리케이션 아키텍처 설계 이후의 과정

❖ 컴포넌트 레파지토리

- 설계된 비즈니스 컴포넌트는 기업 안에서 관리되는 컴포넌트 레파지토리(component repository)에서 설계된 비즈니스 컴포넌트의 사양과 일치하거나 유사한 비즈니스 컴포넌트가 있는지 검색

- 만약 컴포넌트 레파지토리에 비즈니스 컴포넌트가 있다면 해당 컴포넌트를 그대로 또는 약간의 수정을 통해 재사용할 수 있게 된다.

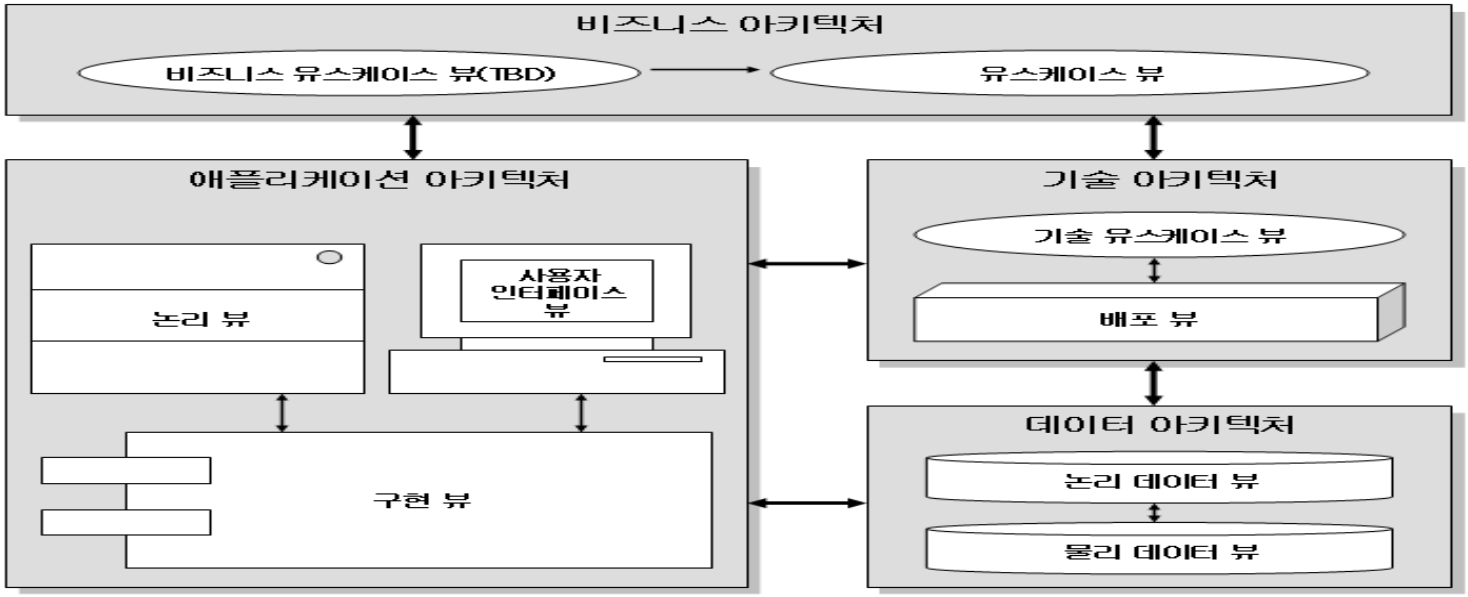
- 이 경우, 컴포넌트 레파지토리에 있는 컴포넌트에 대해서는 설계 단계를 포함하여 그 이후의 과정은 생략될 수 있으며, 이것은 엔터프라이즈 애플리케이션 개발 노력과 기간을 단축시켜주는 결과를 가져오게 함

- 그러나 컴포넌트 레파지토리에 설계된 비즈니스 컴포넌트의 사양과 일치하거나 유사한 비즈니스 컴포넌트가 없다면 설계 단계를 통해 J2EE 등 특정한 기술이 적용된 구현 컴포넌트로 설계되고, 물리적인 구현 컴포넌트로 구현

아키텍처 구조

❖ 개요

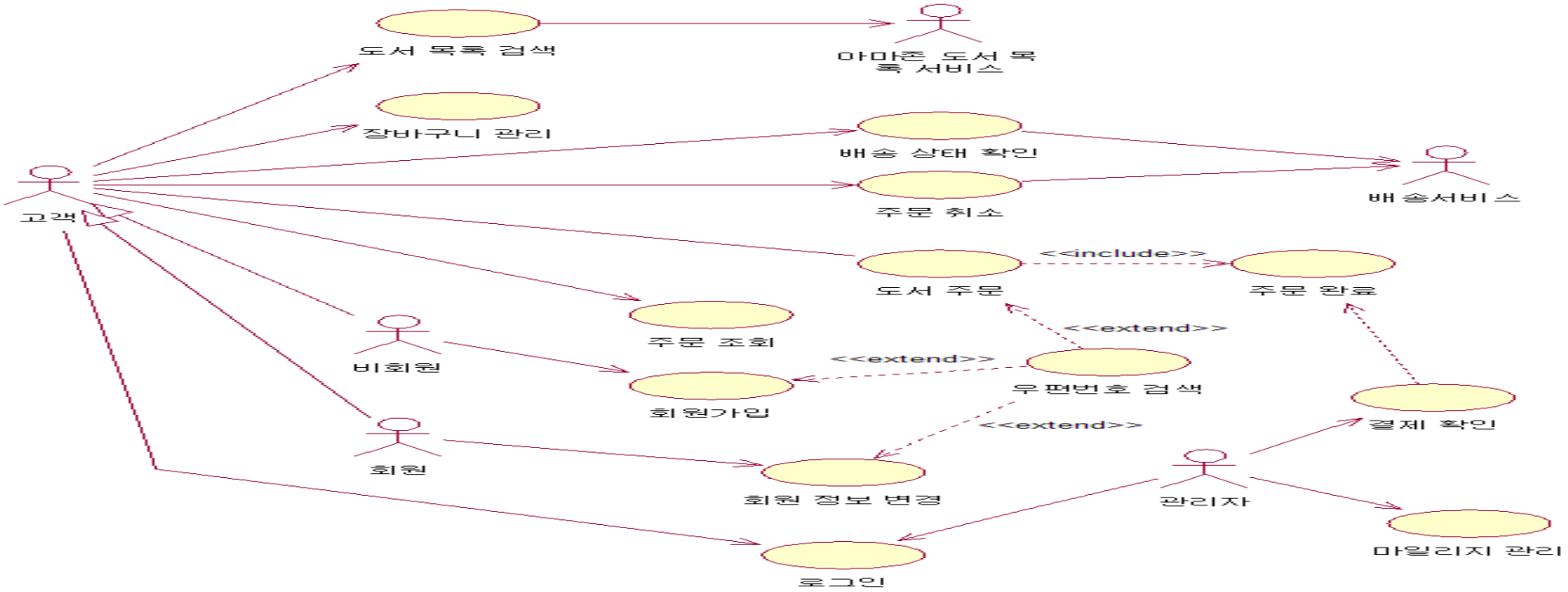
- 아키텍처 구조



아키텍처 구조

❖ 비즈니스 아키텍처

- 도메인 영역에서의 요구사항을 정의하고 구조화하는 것을 지원
- 유스케이스 뷰로 정의하며 요구파악 단계에서 정의된 구조화된 유스케이스 모델
- 비즈니스 아키텍처는 비기능 요구사항에 대한 명세도 포함
- 유스케이스 뷰 - 비즈니스 아키텍처



아키텍처 구조

❖ 애플리케이션 아키텍처

- 도메인 영역에서의 비즈니스 문제를 해결하기 위한 비즈니스 컴포넌트(business component), 사용자 인터페이스 컴포넌트(user interface component), 구현 컴포넌트(implementation component)의 정의

- 애플리케이션 아키텍처는 다음과 같이 세 개의 뷰

  ➤ 논리 뷰(logical view)
  ➤ 사용자 인터페이스 뷰(user interface view)
  ➤ 구현 뷰(implementation view)

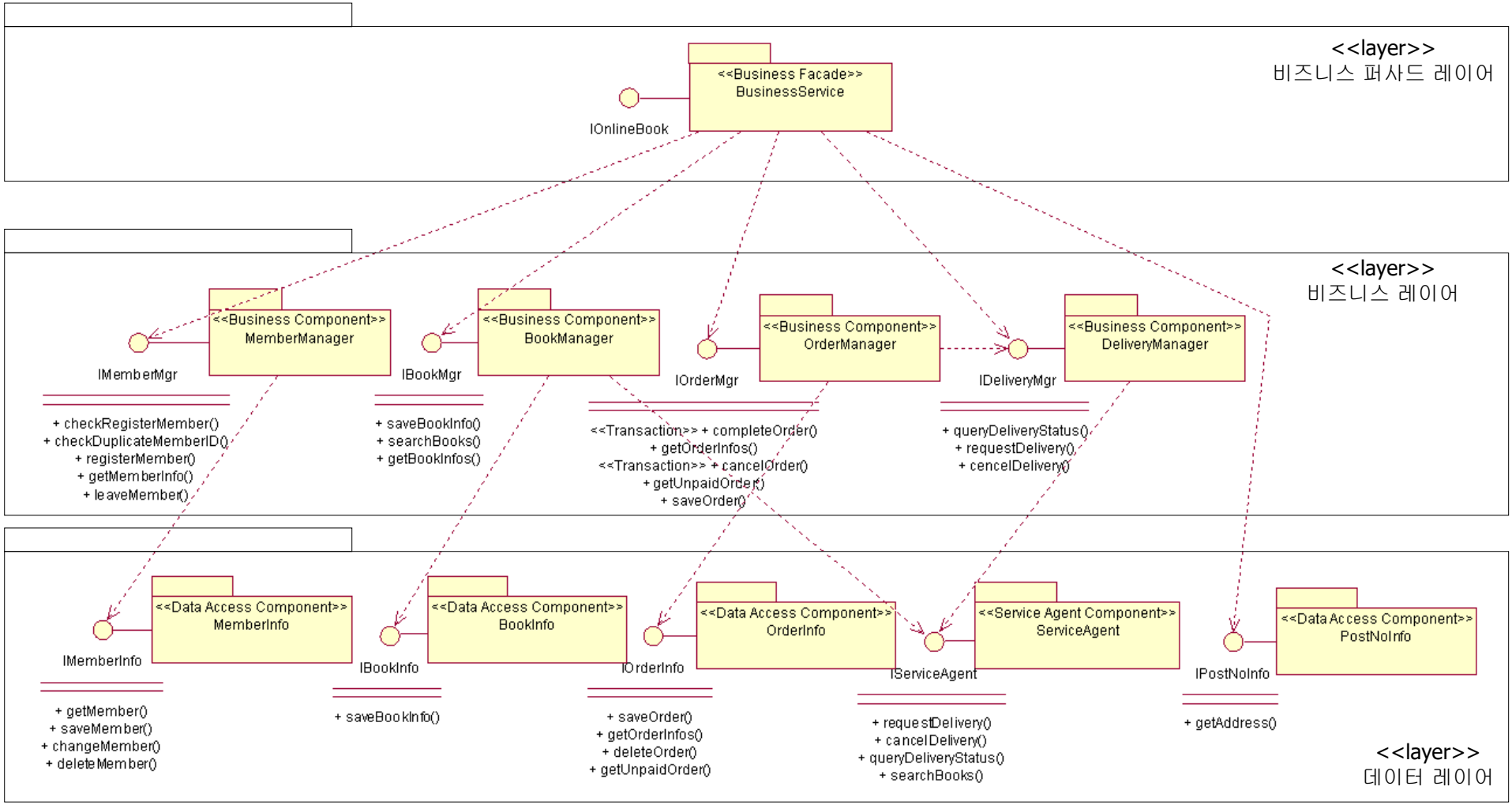
❖ 애플리케이션 아키텍처

- 논리 뷰(logical view)
  - 시스템의 기능적인 요구사항을 지원
  - 논리 뷰의 아키텍처 요소는 비즈니스 컴포넌트
    - 논리 뷰의 아키텍처적인 요소가 개념적인 컴포넌트이어야 할 필요가 있기 때문이며 궁극적으로 이것은 아키텍처적인 요소를 클래스로 정의하는 객체지향적인 방법론과 구별되는 특징
  - 비즈니스 컴포넌트는 시스템의 기능성을 개념적인 비즈니스 인터페이스로 정의하며 구현 뷰에서 정의되는 구현 컴포넌트(implementation component)로 실현
  - 비즈니스 컴포넌트의 논리 모델(logical model)을 클래스 다이어그램(class diagram)으로 표현한다.

아키텍처 구조

❖ 애플리케이션 아키텍처

- 논리 뷰 - 애플리케이션 아키텍처



❖ 애플리케이션 아키텍처(application architecture)

- 구현 뷰(implementation view)

  - 개발 환경 안에서 정적인 소프트웨어 모듈(소스 코드, 데이터 파일, 구현 컴포넌트, 실행 파일)의 구성을 보여줌
  - Applied Software Architecture에서의 모듈 뷰와 코드 뷰가 결합된 것과 유사
  - 아키텍트는 논리 뷰의 아키텍처 요소인 비즈니스 컴포넌트가 현재의 소프트웨어 플랫폼과 기술로 실현될 수 있는지를 보여주어야 한다.
  - 구현 모델(implementation model)을 컴포넌트 다이어그램(component diagram) 또는 클래스 다이어그램(class diagram)으로 표현
  - 구현 모델은 디렉토리 뷰(directory view)와 네임스페이스 뷰(namespace view)로 구분되어 표현

❖ 기술 아키텍처

- 비즈니스 아키텍처에서 도출된 품질 속성을 실현하기 위한 기술적인 기반을 구축하며, 구현 컴포넌트의 실행 환경을 구성
- 컴포넌트 재사용과 요구변화에 대한 견고성을 위주로 프레임워크(framework)를 설계하며, 하드웨어와 네트워크를 포함하는 물리적인 시스템의 구조를 보여줌

- 기술 아키텍처는 크게 다음과 같이 두 개의 뷰로 구성

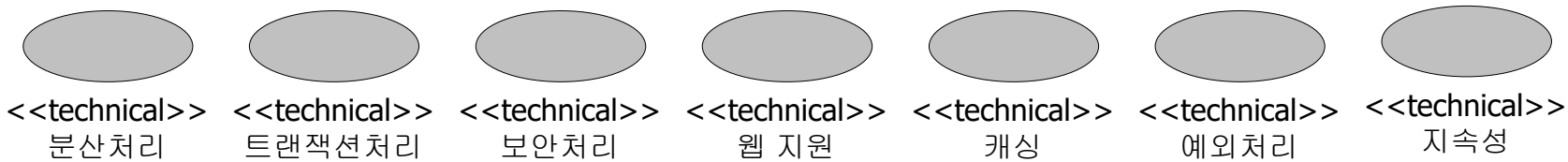
  ➤ 기술 유스케이스 뷰(technical use case view)
  ➤ 배포 뷰(deployment view)

아키텍처 구조

❖ 기술 아키텍처

- 기술 유스케이스 뷰

  - 시스템의 품질 속성에서 기술적인 요구사항을 기술 유스케이스로 정의
  - 기술 유스케이스 실현을 통해 공통 라이브러리 또는 프레임워크를 구성하는 기술 아키텍처 요소를 정의
  - 〈〈technical〉〉 스테리오타입을 갖는 유스케이스로 구성된 유스케이스 다이어그램으로 표현
  - 이와 함께 프레임워크의 설계 모델을 보여주는 클래스 다이어그램(class diagram)도 기술 유스케이스 뷰에 포함될 수 있다.



❖ 기술 아키텍처

- 배포 뷰(deployment view)

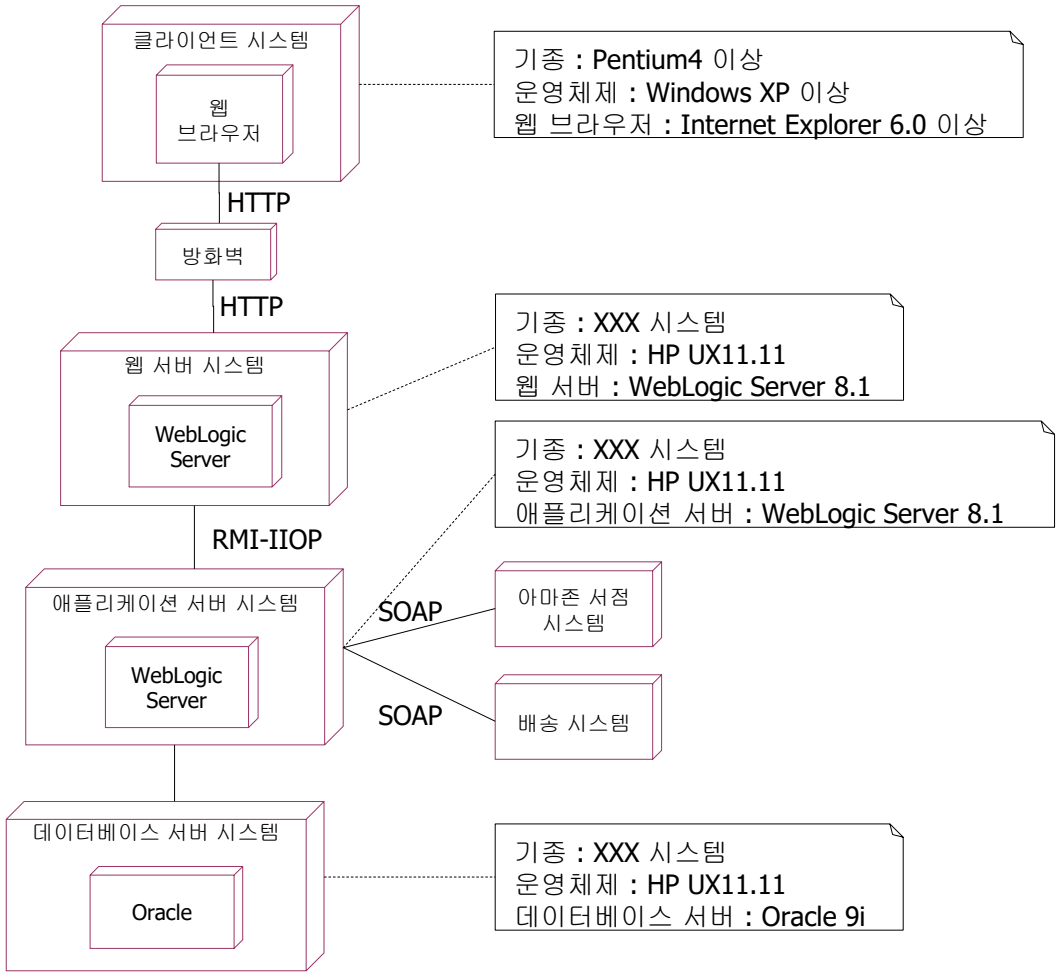
  - 다양한 실행 파일과 다른 런타임 컴포넌트가 해당 플랫폼 또는 컴퓨팅 노드(computing node)에 어떻게 매핑되는가를 보여줌

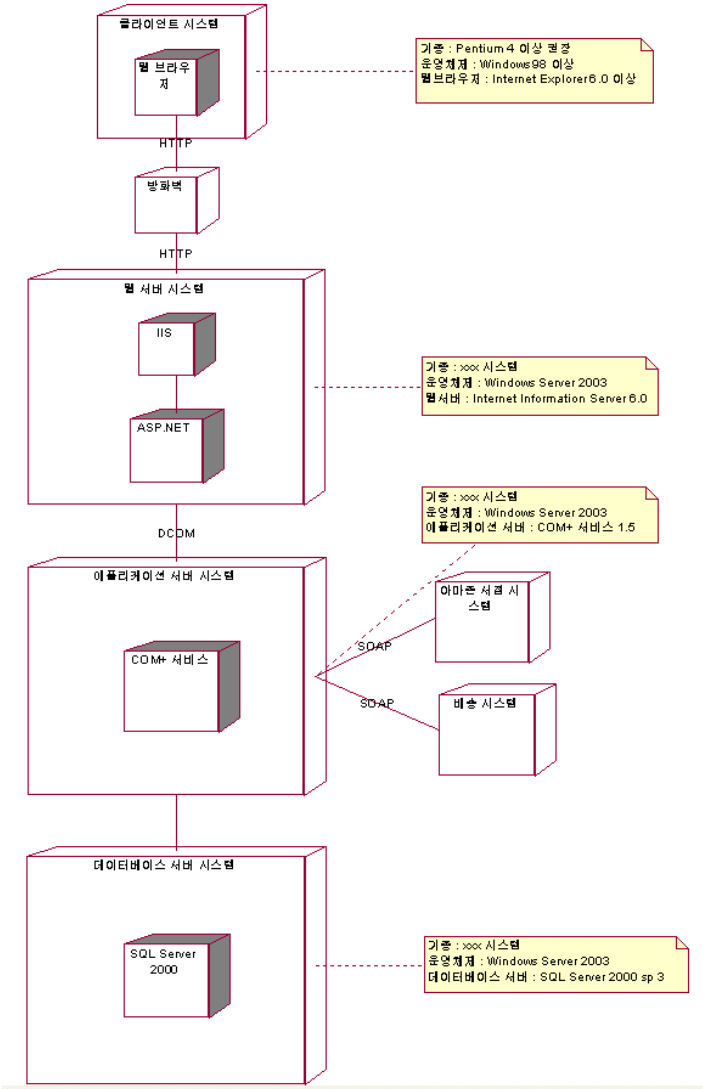
  - 물리적인 노드의 구성과 상호 연결 관계를 배포 다이어그램(deployment diagram)으로 표현

아키텍처 구조

❖ 기술 아키텍처

- 배포 뷰(deployment view) - 기술 아키텍처

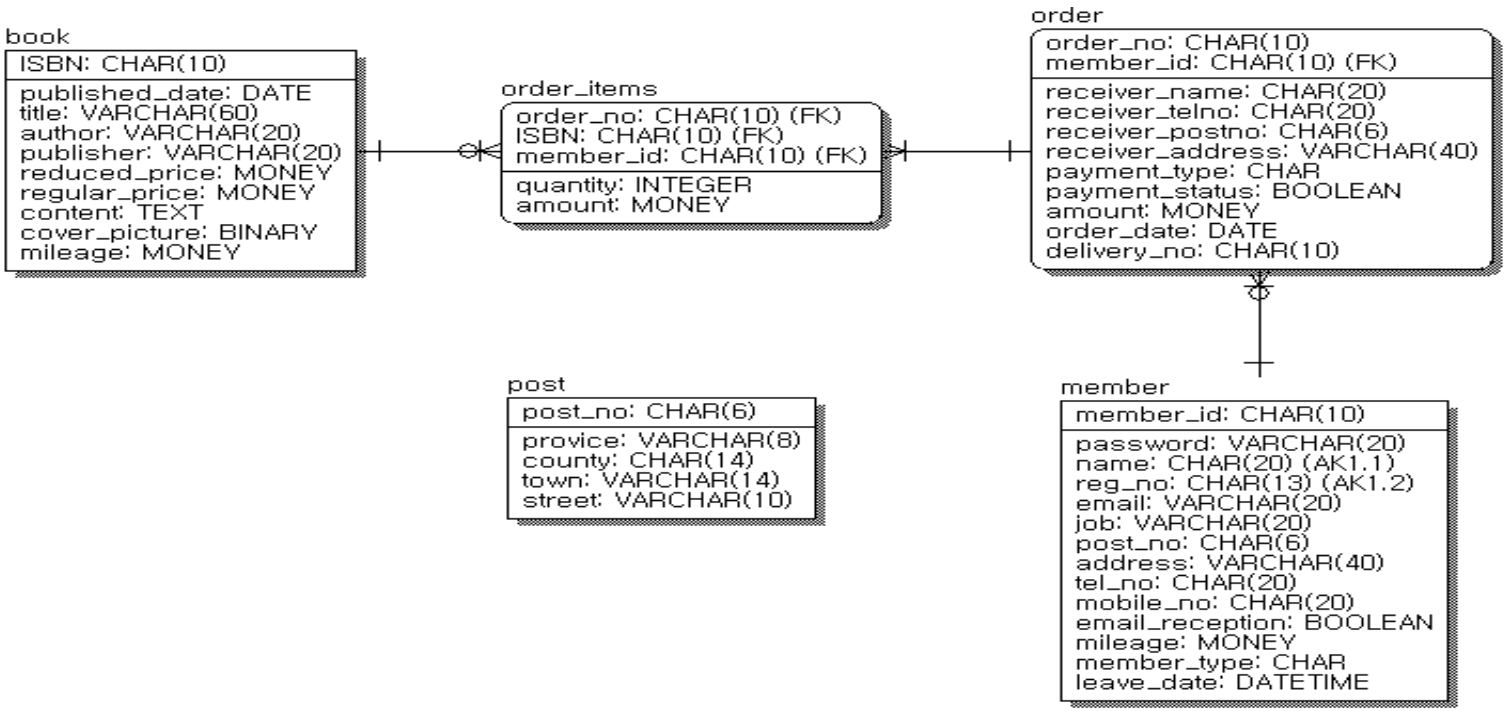




아키텍처 구조

❖ 데이터 아키텍처

- 도메인 영역에서의 정보의 지속성 요구 문제를 처리
- 일반적으로 데이터 뷰(data view)는 E-R 다이어그램(entity relationship diagram)으로 데이터의 논리 모델과 물리 모델을 표현한다. 데이터 아키텍처에 물리 데이터 모델(physical data model)만 포함
- 데이터 뷰 - 데이터 아키텍처



Architecture Design

가용성 설계전술

고가용성의 개요

❖ 고가용성의 정의

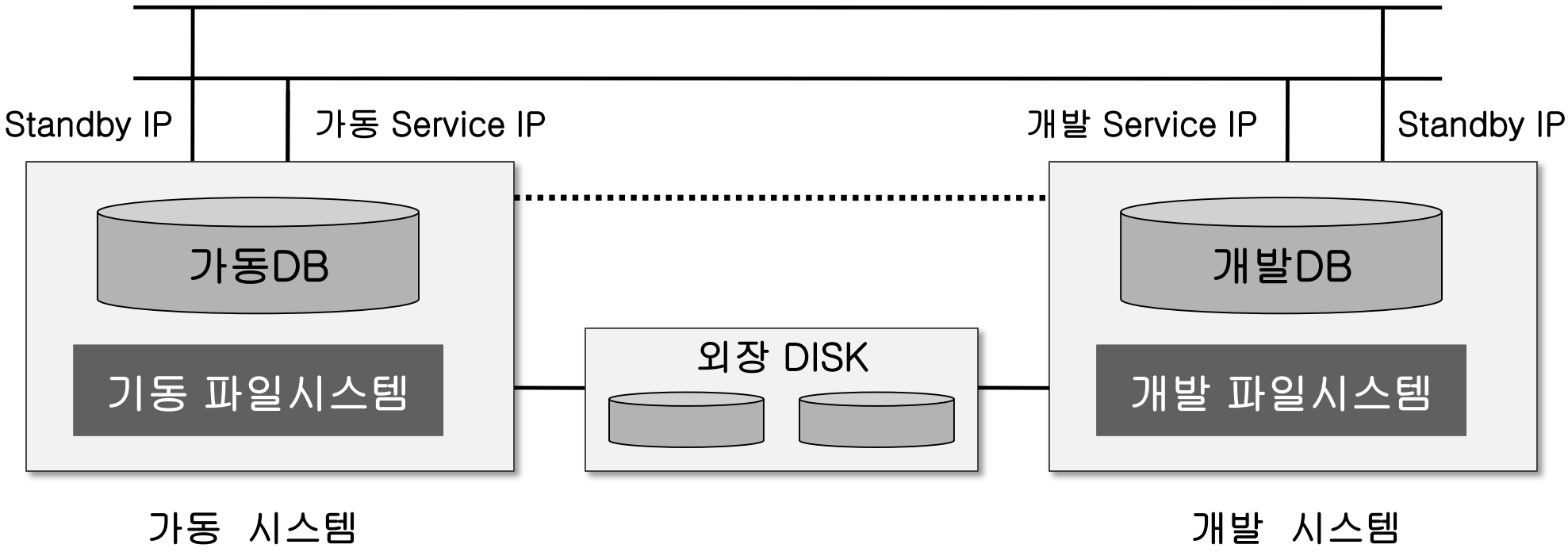
- High Availability 라고 하며 작게는 OS Disk mirroring에서 크게는 시스템 이중화하여 기업의 중요 업무 중단을 최소화하는 기능
- 2대 이상의 시스템을 하나의 Cluster로 묶어서 한 시스템의 장애 발생시 최소한의 서비스 중단을 위해 Cluster내의 다른 시스템이 신속하게 서비스를 Failover하는 기능

❖ HA의 필요성

- 시스템의 고 가용성을 높여 장애 발생 시 업무 서비스의 중단을 최소화
- 기업의 Mission-Critical한 업무에 대한 지속적인 서비스의 필요성 증대

HA의 구성 및 유형

❖ HA 구성도



HA의 구성 및 유형

❖ HA 구성도

- HA 구성에 참여하는 각 시스템은 2개 이상의 N/W Card를 가지면서 N/W통해 상호간의 장애 및 생사 여부를 감시하다
- Standby N/W은 Service N/W 장애 시 백업용으로 사용되고, Private N/W 은HA에 참여하는 시스템들만 통신하는 전용 N/W으로 Heart-Bit N/W 이라고도함
- 외장 Disk는 가동,개발 시스템에서 공유할 수 있어야 하며, Concurrent access또는 순차적인 Access방식에 따라 HA가 다르게 구성된다.

❖ HA 구성유형

- Hot-Standby
  - 가장 단순하면서 많이 사용되는 구성유형
  - 가동 시스템과 평상시 대기 상태 또는 개발시스템으로 운영되는 백업 시스템으로 구성
  - 가동 시스템의 H/W 또는 N/W 장애 발생시 가동시스템의 자원을 백업시스템으로 Failover 하여 가동 업무에 대한 가용성을 보장
  - 외장 Disk는 가동시스템에서만 Access가능, 장애 시에만 백업 시스템이 Access

HA의 구성 및 유형

❖ HA 구성유형

- Mutual Takeover
  - 2개 시스템이 각각의 고유한 가동 업무 서비스를 수행하다가, 한 서버에 장애가 발생되면 상대 시스템의 자원을 Failover하여 동시에 2개의 업무를 수행하는 방식
  - 장애 발생 시 Failover에 대비해 각 시스템의 2개의 업무를 동시에 서비스할 수 있는 시스템 Capacity를 갖추도록 고려해야 함
  - 외장 Disk는 해당 시스템에서만 Access가능함
- Concurrent Access
  - 여러 시스템이 동시에 업무를 나누어 병렬 처리하는 방식으로 HA에 참여하는 시스템 전체라 Active한 상태로 업무를 수행하게 됨
  - 따라서 한 시스템에 장애 발생해도 다른 시스템으로 Failover 하지 않고 가용성을 보장
  - 외장 Disk는 HA참가하는 전체 대상 시스템이 Concurrent Access가능하며, 주로 Database S/W와 쌍을 이루어 구성되는 것이 일반적임
  - 대표적인 예로는 Oracle의 RAC(Real Application Cluster)

HA 동작 방식 및 Fault Tolerant System과 비교

❖ HA 동작 방식

- 한 시스템으로부터 Keep alive packet이 전혀 오지 않는 경우 백업시스템은 상대 시스템이 Down 되었다고 판단하고 이미 정의된 자원(Volume Group, Filesystems, IP Address, Application)을 Failover
- Failover 순서 : 미리 정의된 Scripts에 의해 Network 자원=> Disk자원=> Application 자원 순서로 진행됨

❖ Network Adapter(Card)장애 시

- 가동 시스템 내의 Service Adapter 장애 시 : 시스템에 Service adapter 와 Standby adapter 를 설치 구성한 경우, Service adapter에 장애가 발생하면 Standby adapter가 Service adapter의 IP Address를 Failover
- 가동 시스템 내의 전체 N/W Adapter 장애 시 : 백업 시스템의 Standby Adapter로 가동 시스템의 Service IP Address가 Failover됨.

❖ TCP/IP Network 장애시

- Network 전체장애가 발생하는 경우에는 특별한 동작을 하지 않는다.
- N/W 전체 장애 시에는 백업 시스템으로 Failover해도 동일한 N/W 문제가 존재함

❖ HA와 Fault Tolerant System과의 비교

비교항목	HA	Fault Tolerant
Failover Time	30초 ~ 300초	0초
Concurrent 유지보수	불필요	필수
시스템 비용	2배	10 ~ 20배
Application	거의 제한 없음	제한적
운영체제	범용 OS(UNIX,NT)	독자적 운영체제 사용
사용 H/W 장치	범용 H/W	독자적 특수 H/W

HA 한계점 및 구축 시 고려사항

❖ HA 한계점

- External Disk 자체 장애 발생 시 HA Solution으로 해결 못함
- 장애 발생으로 시스템이 Down되지 않는 경우 자동 Failover가 되지 않음
- 시스템 성능이 저하되는 경우에 자동 감지가 불능
- DB 및 Application 이 Down되는 경우에는 일반적으로 Failover하지 않음
- DB 및 Application 자체 Bug일 경우 Failover가 의미없음
- HA구성에 따른 정보교환으로 시스템의 안정성, 보안성, 성능에 Overhead가 존재함

❖ HA 구축 시 고려사항

- HA 구성 방식 및 대상서버 결정
- 백업 서버 Capacity
- HA 대상 시스템들에 대한 OS 자원 및 사용자 자원 동기화
- 보호될 자원 결정 및 자원 동기화
- External Disk의 보호 방안(2중화 여부)

HA 동향 및 고가용성 시스템 전망

❖ HA의 동향

- 기업의 Mission Critical한 업무에 대해 24X 365 무정지시스템 구성을 위해 시스템 및 Application S/W와 Co-Work하여 Clustering 방식으로 HA를 구성하는 추세임
- HA의 장애 및 가용성 기능에서 발전되어 업무 부하분산 기능까지 포함하는것이 필요함
- Application 시스템 또는 WEB시스템은 가용성을 높이기 위해 여러대의 서버를 별도의 Load Balancer를 통해 장애 및 부하분산을 하는 것이 효과적임
- H/W 관점의 서버 벤더 HA Product와 S/W적인 관점의 Application 밴더의 HA Product기능을 혼합해서 구성하는 것이 대세를 이룰 것으로 판단됨
- 재난복구에 대비해서 데이터 센터내에 Local HA 구성에서 좀 더 넓은 지역으로 femote HA를 구축하는 Solution 활성화가 전망됨

HA 동향 및 고가용성 시스템 전망

❖ 고가용성 시스템의 현황

- HA, FT기반 시스템 및 솔루션 보급의 증가
- MTTR을 줄이기 위한 H/W, S/W적인 연구가 계속 진행중

❖ 고가용성 시스템의 전망

- 비용을 줄이고 FT 수준의 가용도를 구현하기 위해 미들웨어 기반을 솔루션출시
- MMDB, Switch 기술의 발전으로 고가용성 증가가 예상

DRS(Disaster Recovery System)의 개요

❖ DRS(Disaster Recovery System)의 정의

- 재해 발생시 시스템 복구, 데이터 복원 등과 같은 단순 복구차원이 아닌 조직, IT자원, 업무 복원 절차 등을 준비하여, 실제 재해 발생시 기업 비즈니스의 연속성(Continuity)을 보장할 수 있도록 하는 체계

❖ IT관점에서 재해의 의미

- 자연재해 : 홍수, 태풍, 지진, 건물붕괴 등으로 인한 시스템 마비
- 인적재해 : 화재, 폭발, 폭종, 무단점검 등으로 인한 시스템 마비

DRS의 개요

❖ IT관점에서 재해의 의미

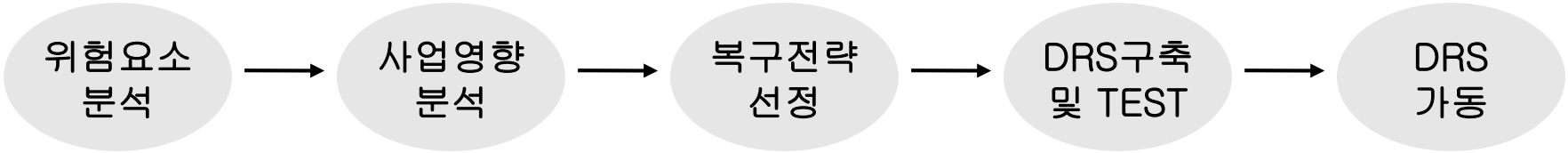
- 장애 : 설비(정전), 전산장비 및 S/W장애, N/W장애 등으로 인한 시스템 마비
- 사고/과실 : 해킹, 거래폭주, 침해사고, 운영자 실수, 절차 오류 등으로 인한 시스템 마비

❖ 금융권 재해복구시스템의 주요기술

- SRDF(Symmertic Remote Data Facility), HRC(Hitachi Remote Copy)
  - 온라인 가동 중에 주 전산시스템 처리와 독립적으로 복구센터에 데이터를 미러링 하는 데이터 이중화 솔루션으로 주 센터의 데이터를 물리적으로 분리된 백업센터의 타겟 디스크에 복제하며, 운영방법은 동기식, 비동기식 등으로 구분
- 동기식(Synchronous)
  - 거래발생시 건별로 주 전산센터에서 백업센터의 정상기록 여부를 확인하고 종료하는 방식으로, 거래량이 많을 경우 백업센터의 처리를 위한 부하로 주 센터의 처리속도 저하 또는 백업센터와의 통신 오류로 인한 주 센터의 영향 초래
- 비동기식(Asynchronous)
  - 주 센터 처리와 백업센터의 처리를 별개의 거래로 처리하는 방식으로 백업센터로 인한 주 센터의 영향은 없으나, 백업센터의 데이터에 대한 정합성을 보장하지 못함

DRS구축 프로세스, 구축 유형 및 구축 방식

❖ DRS(Disaster Decovery System)구축 프로세스



복구전략 선정시 재해 특구 계획
(위기관리 포함) 설정되어야 함

❖ DRS(Disaster Decovery System)구축 유형

구축유형	유형별 주요내용	비 고
Mirror Site	실시간 정보백업(증권,금융) 동일시설, 자원확보 및 동시에 데이터처리 및 운영	적극적 DRS
Hot Site	동일시설, 자원 확보 후 백업체계, 재해 시 대체운영	소극적 DRS
Warm Site	기본시설, 주요자원확보, 간단한 도입 후 복구, 재 가동	
Cold Site	기본시설만 확보, 재해 시 자원도입, 복구, 재 가동	

❖ DRS(Disaster Recovery System)구축 방식

- PTAM(Pickup Truck Access Method
  - 하루 or 일주일 단위로 전체 데이터를 백업 받은 후 원격지에 있는 백업센터로 이송하는 테이프 소산방식

- Log Journaling
  - DBMS Log를 원격지 백업센터로 실시간으로 전송하는 Software방식

- 디스크 미러링
  - S/W방식으로는 서버나 호스트에서 구동되는 별도의 S/W가 데이터 전송을 담당하는 방식으로 이기종 디스크간의 데이터 전송이 가능하다.
  - H/W방식으로는 디스크 장치의 Micro Code(Firmware)가 데이터 전송을 담당하는 방식으로 서버나 호스트의 자원을 사용하지 않는다.

기업에서 DRS(Disaster Recovery System)구축시 고려사항

❖ **주센터와 백업센터간의 거리**
- 솔루션, DRS수준등을 고려하여야 비용산정의 근거가 된다

❖ **데이터 복제 방식**
- 솔루션의 장단점 및 특성을 파악하여 적용하고자 하는 사업의 성격에 따라 구축되어야 한다

❖ **재해복구 목표시간**
- 가장 중요한 요소중 하나로 사업의 특성 및 영향도가 고려되어야 한다

❖ **재해복구 시스템의 가동방식**
- 운영인력을 고려하여 자동, 반자동, 수동등으로 나뉘어 질 수 있다.

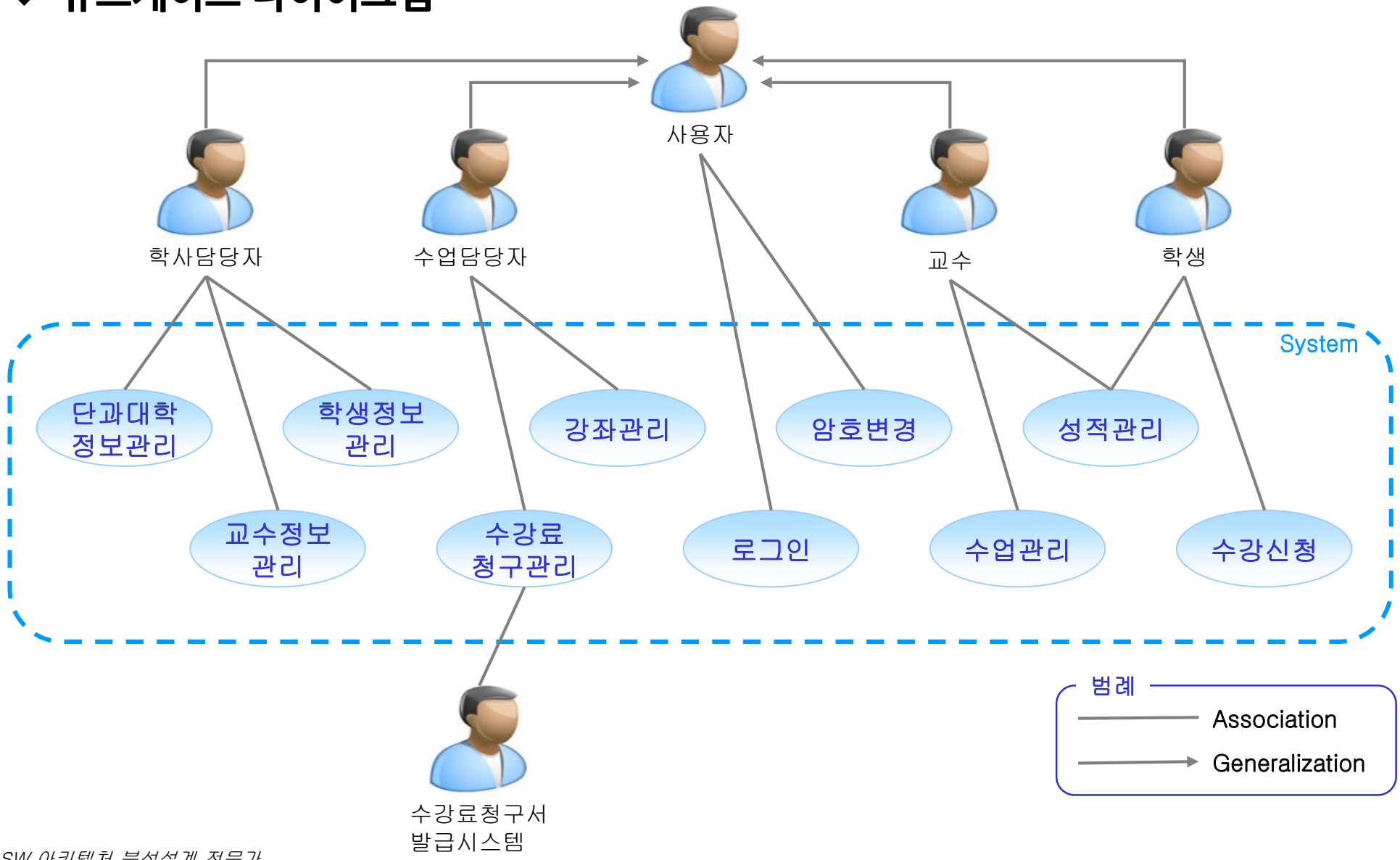
❖ **사용자에 대한 투명성**
- 사업의 성격에 따라 사용자에 인지여부를 고려하여 별도 작업의 필요성을 고려해야 함

Software Architecture

소프트웨어 아키텍처의 실습 사례

요구사항 분석

❖ 유스케이스 다이어그램



❖ 비기능(품질) 요구사항 목록(5개 /총20개)

품질구분	요구사항ID	요구사항명	요구사항설명	난이도	중요도	담당자
성능 Performance	NFP101	수강 신청 기간의 동시접속 보장	수강 신청 기간의 동시 접속 최대 1,000명 보장	상	상	
	NFP201	평균응답 속도	5초이내 응답 보장			
가용성 Availability	NFA101	중단없는 서비스 제공	90% 이상의 가용성 확보	중	상	황**
	NFA201	24시간내 복구	원격지 사용한 백업 시스템			
	NFA301	알림기능	장애발생시 관리자에게 SMS 보내는 기능			
변경가능성 Modifiability	NFM101	기능추가 용이성	유지보수 및 변경이 쉬워야함	중	상	서**
	NFM201	SMS 서비스 연계	SMS 서비스 업체와 연계하여 수강신청관련사항,성적사항,출결사항,등록금청구,각 종 교내공지사항 등의 SMS 서비스를 학생과 교수에게 제공한다.	중	중	오**
	NFM301	시스템 자원 변경 유연성	시스템 품질을 향상시키기 위해 하드웨어등의 증설이 쉬워야함.(유지보수 및 업그레이드 쉽게)			
보안 Security	NFS101	웹서버내 보안인증서 탑재	웹서버에 보안인증서를 탑재하여 SSL 구축			
	NFS201	개인정보	개인정보 보호를 위한 DB암호화 적용	중	중	박**
	NFS301	인증강화를 위해 인증서 사용	회원가입시 Two-Factor 인증 사용(비밀번호와 인증서를 사용한 인증시스템 구축)			
	NFS401	접근권한관리 체계 구축	사용자별 접근할수 있는 권한을 제공해야함.	상	상	서**
	NFS501	접근이력 기록	중요데이터 접근이나 변경시 로그 남기기			

❖ 비기능(품질) 요구사항 목록(5개 /총20개)

품질구분	요구사항 ID	요구사항명	요구사항설명	난이도	중요도	담당자
보안 Security	NFS101	비밀번호 정책	영문자 숫자 특수문자 포함 7자리로 하고, 6개월 마다 한 번씩 교체를 알리고, 바로 직전에 사용한 비밀번호로는 교체 불가			
시험가능성 Testability	NFT101	단위테스트 제공	모듈별로 테스트 가능하도록 단위테스트 케이스 제공	중	중	박**
	NFT201	테스트 자동화	테스트 장비에서 자동화 테스트를 수행한 후 운영서버에 반영			
사용성 Usability	NFU101	다양한 브라우저 지원	다양한 브라우저 지원 가능	중	상	황**
	NFU201	실시간 관리자화면	수강신청 현황 등의 현재 진행 상황을 실시간으로 확인가능한 화면제공	중	중	이**
	NFU301	SSO 사용	SSO 사용하여 편리한 로그인 기능 수행			
	NFU401	요청처리의 상태확인 표시	요청후 응답 받을때 까지 상태 표현			

❖ 가용성 품질 시나리오

요구사항	중단 없는 서비스 제공
자극의 근원	하드웨어 및 네트워크
자 극	장애
대 상	APP 서버장비 및 네트워크 장비
환 경	정상작동
응 답	운영자에게 통지(SMS, e-mail, 사내 비상경보음 작동) 서비스가 중단되지 않고 정상적으로 서비스가 되야함.
응담측정	하드웨어 및 네트워크 장애로 인한 서비스 정지 시간(0초)
비 고	

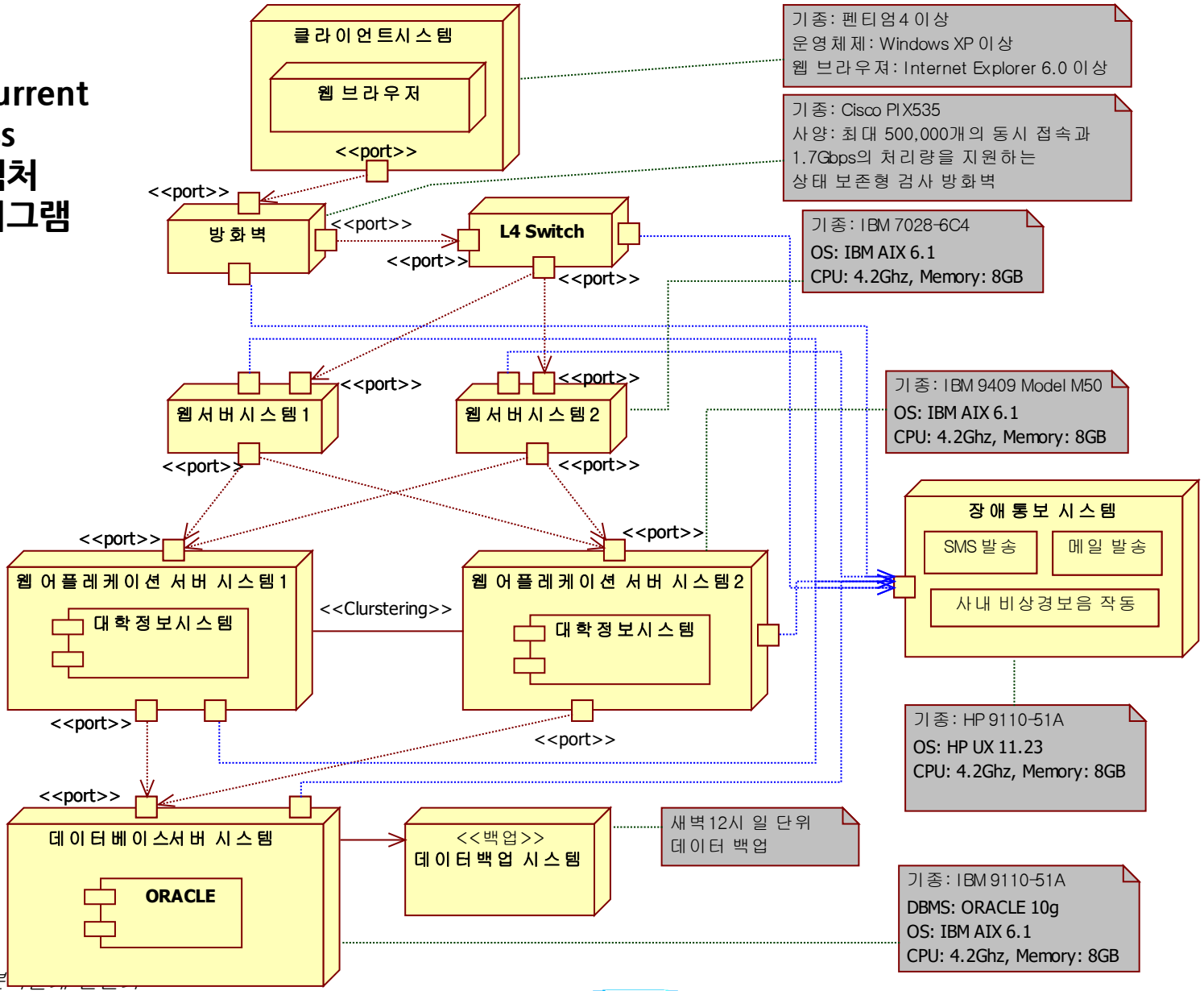
아키텍처 접근 방법 분석

❖ 가용성 품질 시나리오

No	아키텍처 의사결정	위험요소	민감도	절충점
1	Concurrent Access로 시스템을 구성하여 정상적인 서비스를 제공한다.	R1, R2, R3	S1, S3, S4	T1
2	TCP/IP Network 장애 시 외부 보조 네트워크 및APP서버를 이용하여 정상적인 서비스를 제공한다.	R1, R2, R3	S1, S2, S3, S4	T1

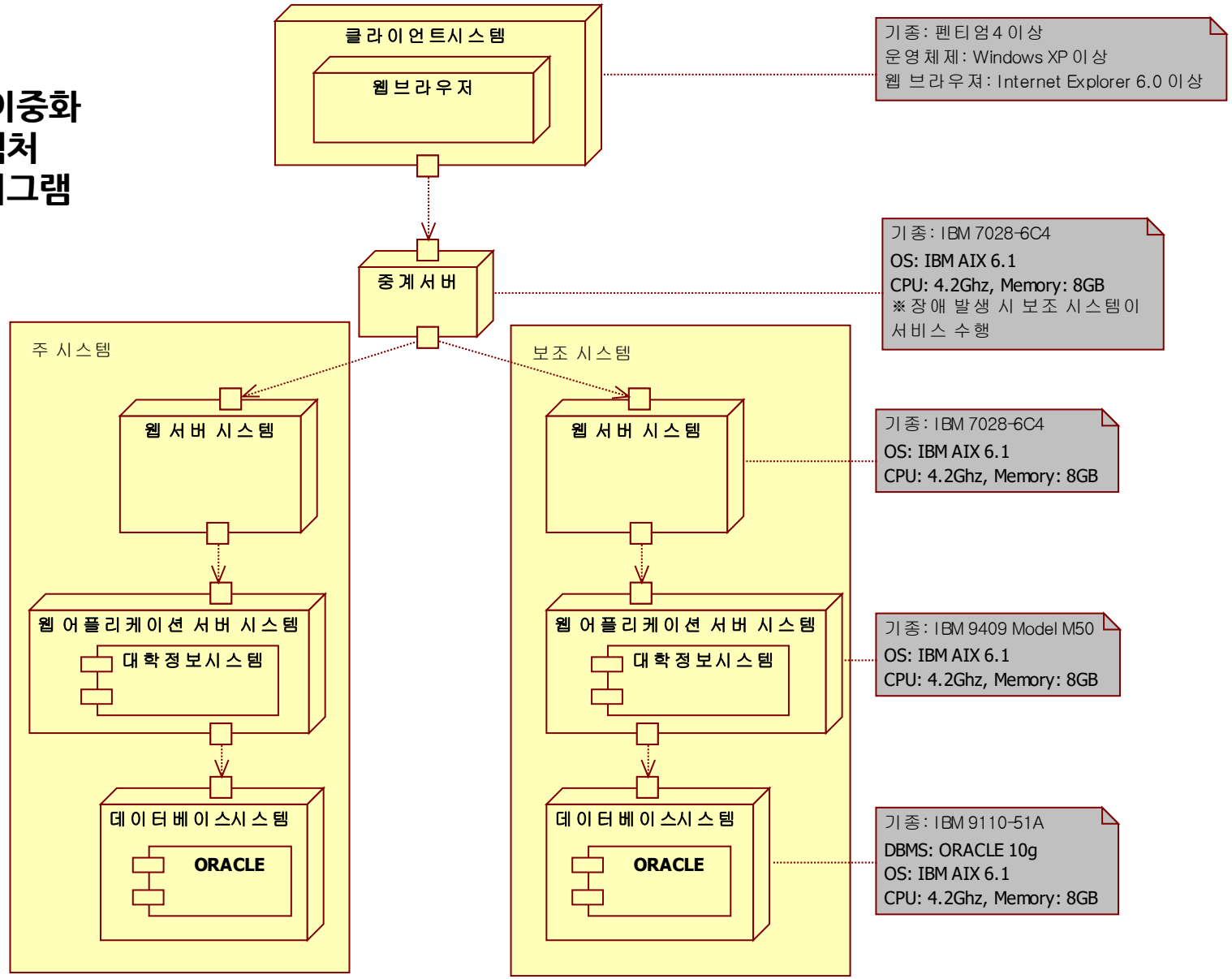
아키텍처 접근 방법 분석

❖ 1안) Concurrent Access 아키텍처 다이어그램



아키텍처 접근 방법 분석

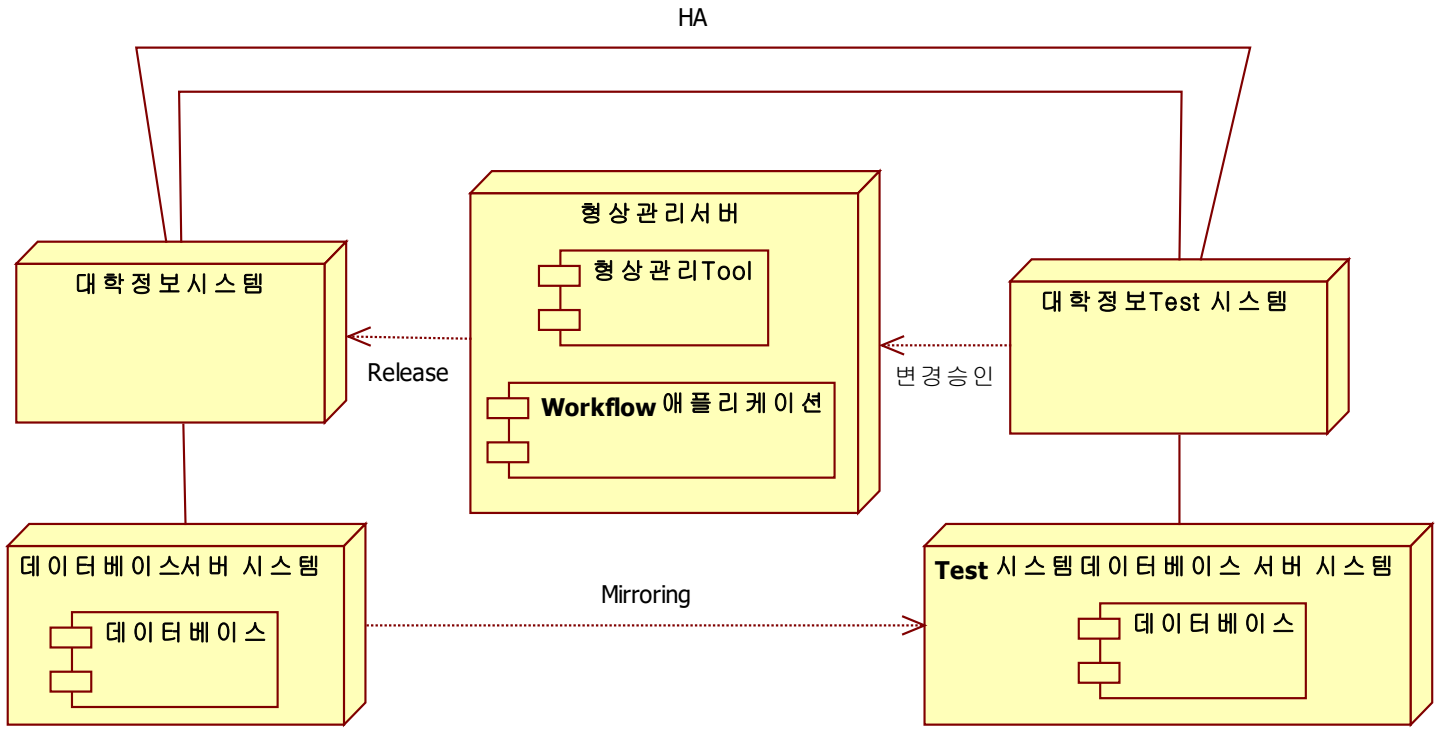
❖ 2안)
서버 이중화
아키텍처
다이어그램



아키텍처 접근 방법 분석

❖ 배포뷰

- Test 서버를 실 서버와 HA로 구성



SW 아키텍처 전략 수립

품질구분	요구사항ID	비기능 요구사항	아키텍처 설계
성능 Performance	NFP102	평균응답속도	접속자수 1,000명 기준 G/W까지 최대 5초 이내 응답 보장
가용성 Availability	NFA102	데이터 유실 방지	스토리지 또는 데이터베이스 시스템의 장애로 인해 서비스가 중지되어 데이터 유실이 발생했을 경우 가장 최근의 데이터로 복원하여 3시간 내 서비스를 가동한다
변경가능성 Modifiability	NFM103	개발기능 컴포넌트화	응집도,결합도의 최적화 고려한다. 컴포넌트는 상호 독립적이어야 한다.
보안 Security	NFS101	로그인	ID/PW & 인증서 2Factor 이상 인증 수행
시험가능성 Testability	NFT104	변경 오류 최소화	기능 요구 변경에 따른 변경 오류 없음
사용성 Usability	NFU102	요청 진행정보 표시	사용자 요청 프로세스의 진행 상황 표시

❖ 배포뷰