컴포넌트 기반 설계/
소프트웨어 아키텍처 설계

한국기술교육대학교
온라인평생교육원

학습내용

- 컴포넌트 기반 개발의 개요
- 컴포넌트 기반 설계
- 소프트웨어 아키텍처 설계

학습목표

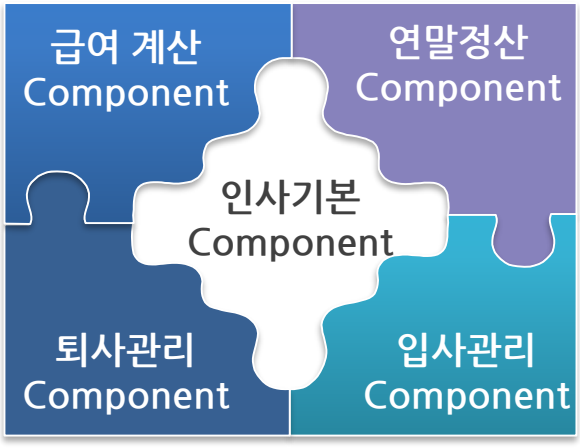
- 소프트웨어 컴포넌트 및 컴포넌트 기반 개발 방법론의 개념과 특징을 설명할 수 있다.
- 컴포넌트 식별, 추출, 명세 등 컴포넌트 기반 설계의 주요 개념을 설명할 수 있다.
- 소프트웨어 아키텍처에 대한 이해를 통해 소프트웨어 아키텍처 패턴을 설계할 수 있다.

컴포넌트 기반 개발의 개요

소프트웨어 컴포넌트(Software Component)

개념

- 특정한 기능을 수행하기 위해 독립적으로 개발되고 잘 정의된 인터페이스를 가지며 다른 부품과 조립되어 애플리케이션을 구축하기 위해 사용되는 소프트웨어 부품



〈 인사관리 애플리케이션 〉

컴포넌트 기반 개발의 개요

소프트웨어 컴포넌트(Software Component)

특징

- 독립적인 소프트웨어 모듈
- 구현, 명세화, 패키지화 및 배포 가능
- 하나의 컴포넌트는 하나 이상의 클래스들로 구성
- 컴포넌트는 반드시 인터페이스를 통해서만 접근 가능
- 실행코드 기반 재사용

컴포넌트 기반 개발의 개요

소프트웨어 컴포넌트(Software Component)

유형

분산 객체 컴포넌트

- EJB, CORBA, COM+ 등 분산 객체 환경 지원 컴포넌트

비즈니스 컴포넌트

- 물리적으로 배포할 수 있는 독립된 하나의 비즈니스 개념을 구현한 컴포넌트

확장 비즈니스 컴포넌트

- 확장을 고려하여 설계된 비즈니스 컴포넌트의 집합
- 그룹 형태로 재사용이 가능한 항목의 집합체

시스템 컴포넌트

- 비즈니스 가치를 제공하기 위해 같은 일을 하는 시스템 수준의 컴포넌트들의 집합

컴포넌트 기반 개발의 개요

컴포넌트 기반 개발(Component Based Software Development)

개념

- 소프트웨어 개발 주기의 모든 단계에서
  컴포넌트를 소프트웨어 부품처럼 미리 만들어서
  애플리케이션을 조립하듯이 만들어나가는 개발 방법론

개발 절차

CD(Component Development)

- 도메인 분석을 통하여 컴포넌트를 추출한 뒤 이를 설계 및 구현한 후 컴포넌트 Repository에 등록을 하여 향후 재사용할 수 있도록 하는 단계

CBD(Component Based Development)

- 컴포넌트 Repository를 통하여 재사용할 컴포넌트를 검색·조립하여 애플리케이션을 구축하는 단계

컴포넌트 기반 개발의 개요

컴포넌트 기반 개발(Component Based Software Development)

특징

1 생산성 향상

- 부품 조립을 통한 시간 단축 및 애플리케이션 개발 시간 단축
- 개발자의 생산성 향상
- 품질이 검증된 컴포넌트 사용

2 고품질의 애플리케이션 구축 가능

- 지속적인 품질 관리로 검증된 컴포넌트의 사용
- 품질을 고려한 컴포넌트 설계 및 구현

컴포넌트 기반 개발의 개요

컴포넌트 기반 개발(Component Based Software Development)

특징

3 재사용·대체성

- 실행 기반의 재사용
  - ➡ 동일 기능의 중복 개발 없이 기존에 개발된 컴포넌트를 이용하여 새로운 시스템 구축 가능
- 모델과 프레임워크 기반의 재사용
  - ➡ 다른 컴포넌트에 영향, 시스템 전체 변경 없이 새로운 기능 추가 및 컴포넌트 변경(대체) 가능

4 변경 용이성

- 요구사항의 변화와 수용에 안정적이고 신속한 변경 가능
- 업무 변경에 따른 위험 최소

컴포넌트 기반 개발의 개요

컴포넌트 기반 개발(Component Based Software Development)

특징

5 기술 집약성

- 기술 숙련에 대한 집중
- 아키텍처, 프레임워크, 분산 객체 기술 등

6 관리 용이성

- 독립적인 컴포넌트 단위의 관리로 복잡성 최소
- 제작 주기에 대한 예측 가능
- 제품 외주화 및 구매에 대한 선택 기회 부여

7 사용자 중심

- 사용자 관점 요구사항 분석으로 컴포넌트 식별 가능
- 사용자 중심의 개발로 사용자 만족도 증가

컴포넌트 기반 설계

컴포넌트 식별

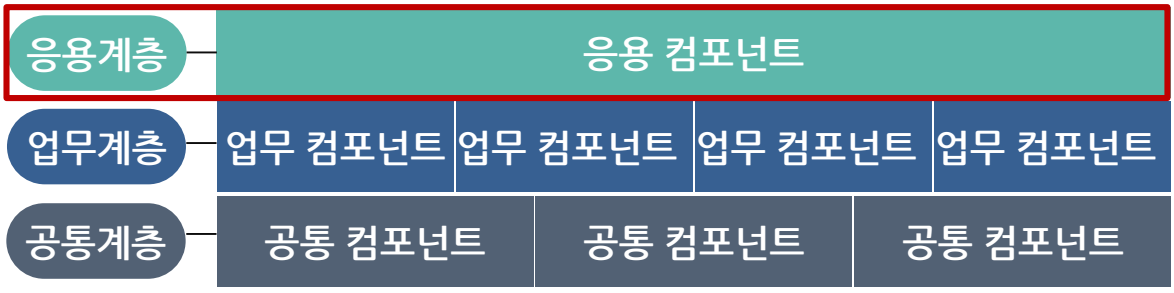
기법

- 유스케이스(Use Case)와 클래스(Class) 상관분석을 통하여 컴포넌트 식별
- 컴포넌트를 응용·업무·공통 컴포넌트로 구분하여 추출

컴포넌트 기반 설계

컴포넌트 식별

비즈니스 컴포넌트의 계층구조

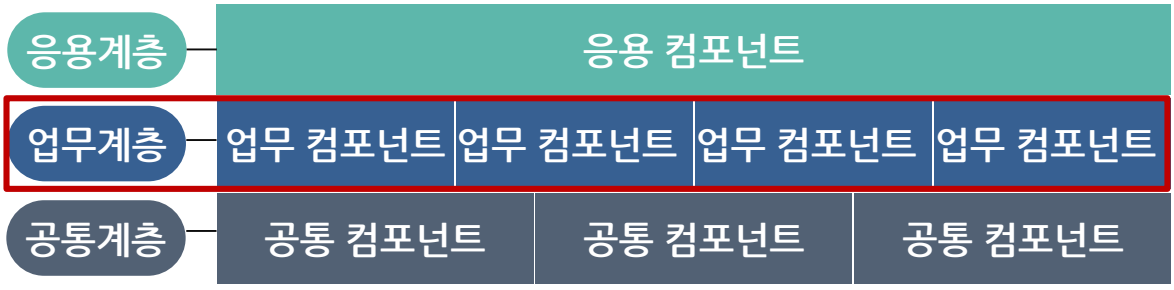


구분	관련컴포넌트	설명
참조 유스케이스	응용 컴포넌트	참조 유스케이스 (조회 전용 유스케이스)는 응용 컴포넌트 추출 대상

컴포넌트 기반 설계

컴포넌트 식별

비즈니스 컴포넌트의 계층구조

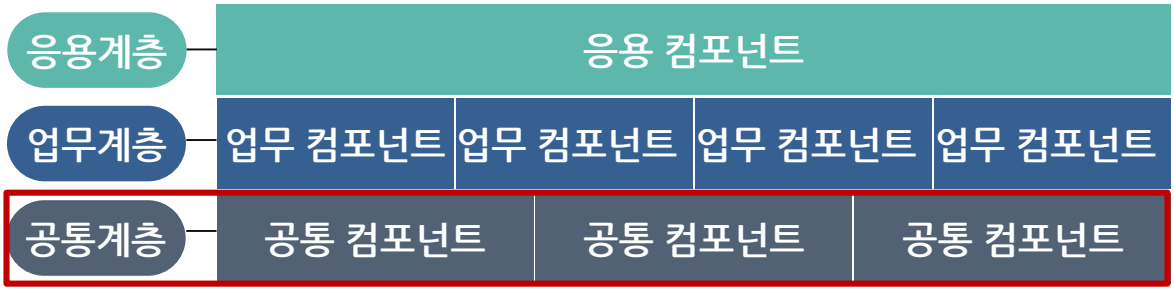


구분	관련컴포넌트	설명
참조 및 공통 유스케이스 이외 유스케이스	업무 컴포넌트	참조 및 공통 유스케이스 이외의 유스케이스는 업무 컴포넌트 추출 대상

컴포넌트 기반 설계

컴포넌트 식별

비즈니스 컴포넌트의 계층구조

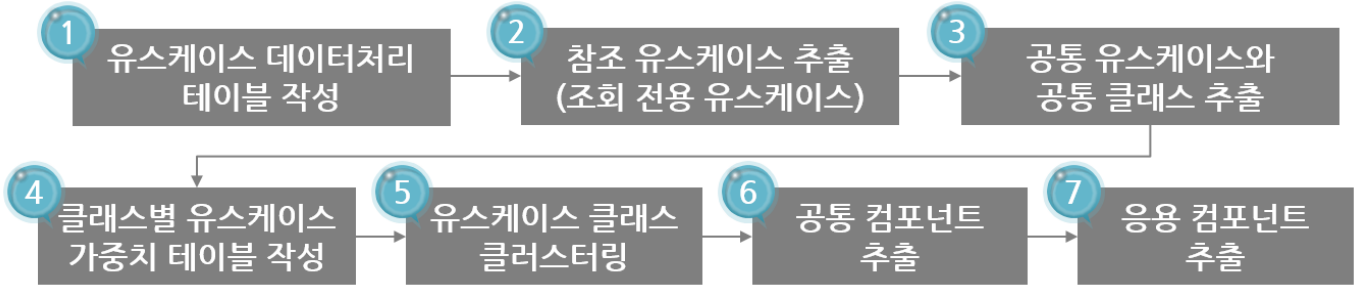


구분	관련컴포넌트	설명
공통 유스케이스 및 클래스	공통 컴포넌트	공통 유스케이스는 유스케이스 Diagram의 <<include>>, <<extend>>인 유스케이스

컴포넌트 기반 설계

컴포넌트 식별

컴포넌트 추출 절차



참조 유스케이스 추출

- 추출할 유형 : 응용 컴포넌트
- 참조 유스케이스(조회 전용 유스케이스)는 응용 컴포넌트 추출 대상

공통 유스케이스와 공통 클래스

- 추출할 유형 : 공통 컴포넌트
- 공통 유스케이스 : 유스케이스 Diagram의 <<include>>, <<extend>>인 유스케이스

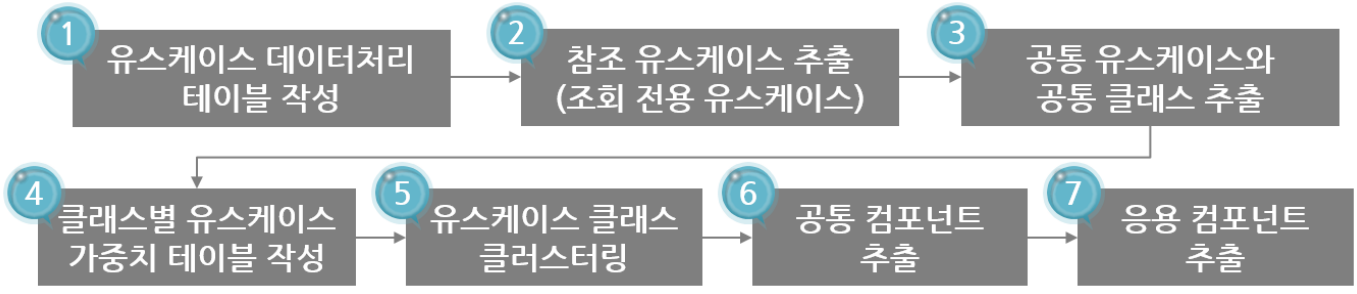
참조 유스케이스와 공통 유스케이스 이외의 유스케이스

- 추출할 유형 : 업무 컴포넌트
- 조회 전용 유스케이스와 공통 유스케이스 외의 유스케이스는 업무 컴포넌트 추출 대상

컴포넌트 기반 설계

컴포넌트 식별

컴포넌트 추출 절차



클러스터링을 통한 업무 컴포넌트 후보 도출

- 추출할 유형 : 업무 컴포넌트
- 참조 및 공통 유스케이스 이외의 유스케이스로 구성된 '클래스별 유스케이스 가중치 테이블'을 작성하여 추출

공통 컴포넌트 추출

- 추출할 유형 : 공통 컴포넌트
- 공통 유스케이스로만 구성된 '클래스 유스케이스 가중치 테이블'을 작성하여 추출

컴포넌트 기반 설계

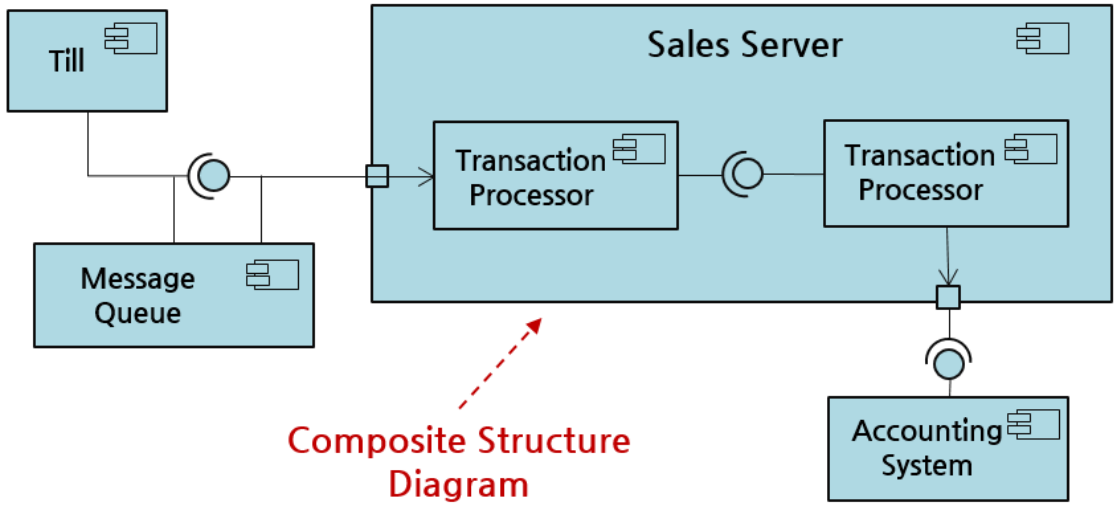
컴포넌트 명세

컴포넌트 명세의 구분

구조 명세 | 기능 명세

구조 명세

- 공통 컴포넌트를 구성하는 각각의 클래스들의 구조와 이들 간의 관계를 UML의 컴포넌트 다이어그램으로 설계하는 것
- 현재의 컴포넌트 구성도를 나타냄



〈 An Example Component Diagram 〉

컴포넌트 기반 설계

컴포넌트 명세

컴포넌트 명세의 구분

구조 명세 | 기능 명세

기능 명세

- 구조 명세된 컴포넌트들의 기능을 자세히 기술하는 것
- 컴포넌트가 수행하는 프로세스를 명세함
- 컴포넌트 ID, 컴포넌트명, 내부 클래스명, 외부 인터페이스명 등을 기술함

컴포넌트 ID	SS_CO_010	컴포넌트명	원화자금이체
컴포넌트 개요	망 참가기관이 개설된 당좌 예금계좌를 통한 기관 내 본/지점 간 자금 거래 결제 및 제한된 범위 내 기관 간 자금 거래의 결제 처리		
내부 클래스			
ID	클래스명	비고	
SS_IC_010	Cteba001SBBean	구현된 메소드는 원화자금이체 신청, 수취인 지정자금이체 신청, 미결제지정시점 예약자금이체 취소신청 업무를 수행한다.	
SS_DC_010	TransManager	Browse 오퍼레이션을 호출하고 처리된 결과를 DocListDTO로 리턴한다.	
SS_IC_020	Ctrba002SBBean	호스트조회를 위한 hostsystem 컴포넌트를 lookup 및 processSend 오퍼레이션을 호출하고 처리결과를 리턴한다.	
인터페이스 클래스			
ID	인터페이스명	오퍼레이션명	구분
SS_IC_030	HS_IF11110	getEventList	Serviced
		…….	……

소프트웨어 아키텍처 설계

소프트웨어 아키텍처

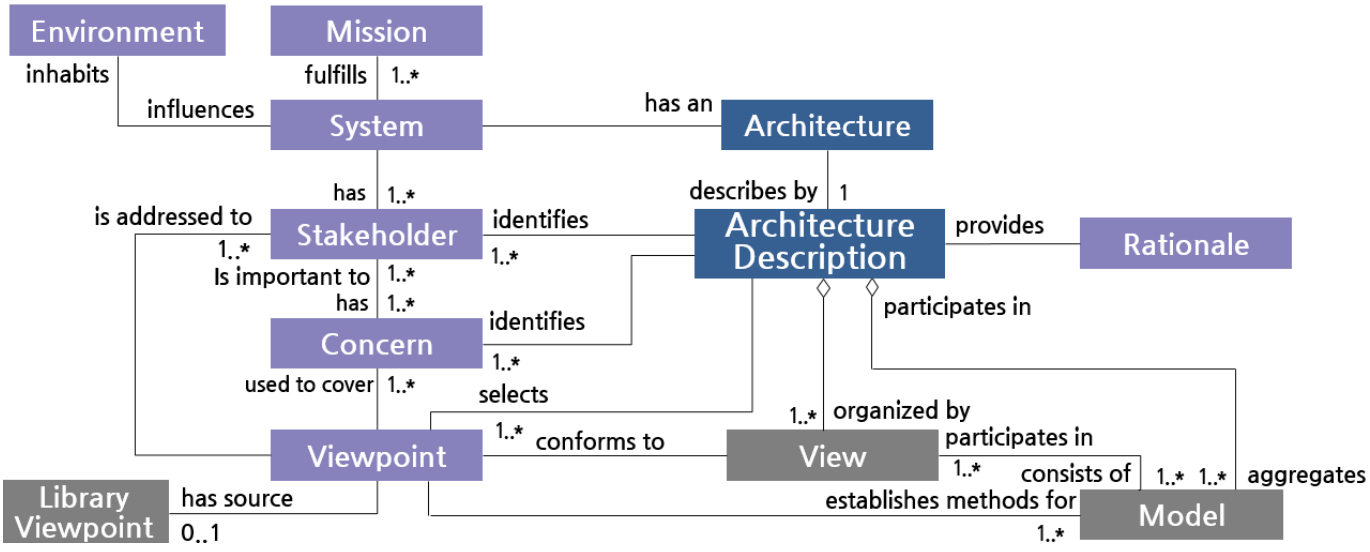
정의

- 소프트웨어를 구성하는 컴포넌트와 이들 간의 상호작용 및 관계
- 이들이 구성하는 소프트웨어의 설계 및 진화를 위한 원칙들의 집합

소프트웨어 아키텍처 설계

소프트웨어 아키텍처

국제 표준(IEEE 1471)

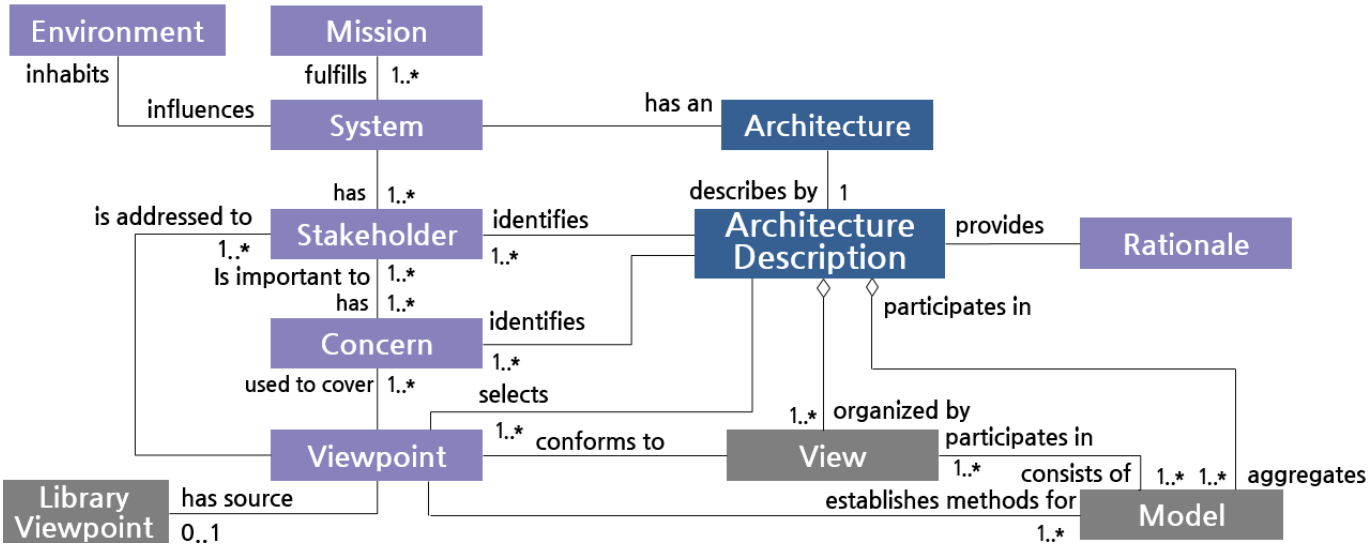


항목	설명
시스템 (System)	• 비즈니스 목적이나 사명(Mission) 완수, 특정 환경(Environment)의 영향, 하나의 아키텍처(Architecture) 소유 • 특정기능이나 기능 세트를 달성하기 위해 조직된 컴포넌트 집합
이해관계자 (Stakeholder)	• 시스템에 관심(Concern)을 갖는 사람이나 조직
관점 (Viewpoint)	• 모델(Model) 작성 방법을 정의 • 신규 생성 또는 라이브러리 관점(Library Viewpoint) 활용

소프트웨어 아키텍처 설계

소프트웨어 아키텍처

국제 표준(IEEE 1471)

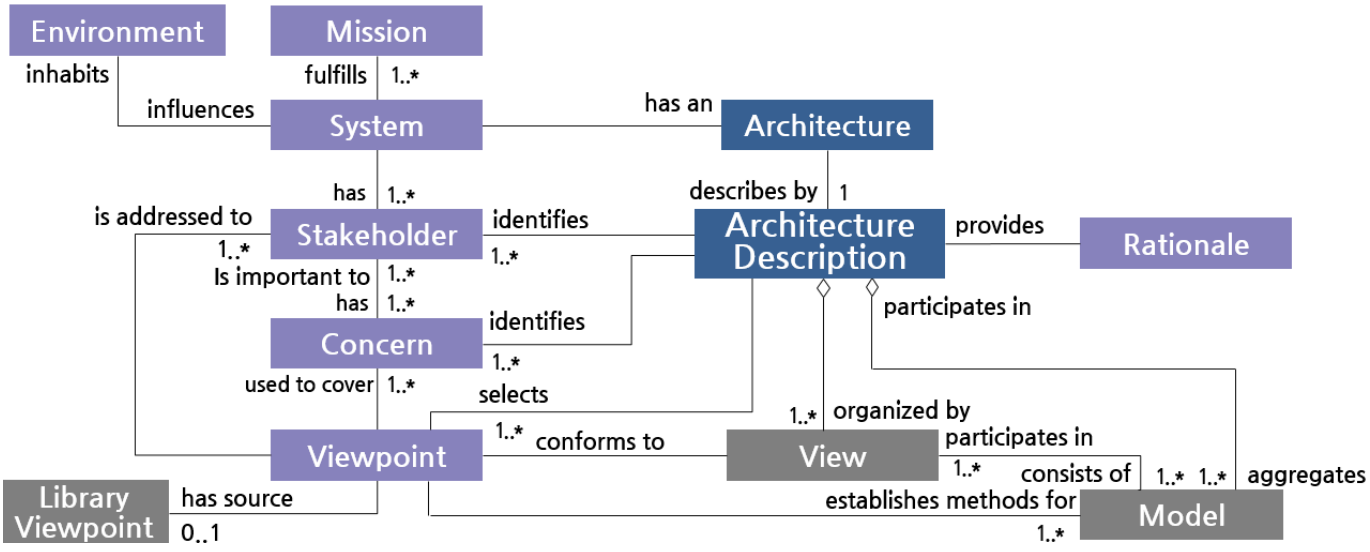


항목	설명
아키텍처	• 아키텍처 기술서로 문서화되어 구체화 • 아키텍처 결정 근거(Rationale) 제시 필수
아키텍처 기술서	• 이해관계자들의 시스템에 대한 관심을 관점(Viewpoint)에 맞춰 작성한 뷰(View)로 구성
결정 근거 (Rationale)	• 여러 이해당사자가 불필요한 논쟁 감소, 의사소통 원활 • 아키텍처 평가 시 주요 판단 기준

소프트웨어 아키텍처 설계

소프트웨어 아키텍처

국제 표준(IEEE 1471)



항목	설명
관심 (Concern)	• 각각의 이해관계자가 가지고 있는 관심사항
환경 (Environment)	• 시스템에 대한 개발, 작동, 정책, 기타 영향요소들의 설정과 환경 • 내부제약조건(조직 내 표준 등), 외부제약조건(법규, 제도), HW시스템
사명 (Mission)	• Stakeholder의 목적 달성을 위해 시스템이 수행하는 연산

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

정의

- 문제를 해결하는 해법으로 소프트웨어 시스템의 기본구조와 관련된 것을 다룰 경우 아키텍처 수준의 패턴이라고 함

디자인 패턴과 이디엄

디자인 패턴

- 소프트웨어 시스템의 서브시스템이나 컴포넌트들 혹은 그것들 간의 관계를 해법으로 사용하는 경우

이디엄

- 특정 프로그래밍 언어의 기능을 이용하여 컴포넌트들 혹은 컴포넌트들 간 관계의 특정 측면을 구현하는 방법

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

종류

- Layer 패턴
- Blackboard 패턴
- Broker 패턴
- MVC 패턴
- Publisher-Subscriber 패턴
- Pipes and Filters 패턴

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

종류

Layer 패턴

- 가장 일반적으로 사용하는 아키텍처 패턴
- Subtask들을 그룹으로 묶어 사용 허가 관계를 표시하는 패턴
- 모듈의 재사용성을 높여 유지보수성이나 이식성에 좋음

Blackboard 패턴

- Shared Data, Database와 같은 데이터 중심 패턴 중 하나임
- 명확히 정의된 문제 해법이 없을 때 문제를 풀어가는 하나의 방식을 정의한 패턴

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

종류

Broker 패턴

- 외부에 분산된 컴포넌트를 호출하려고 할 때 클라이언트 Request를 분석하여 서버 컴포넌트에 전달하고 그 결과값을 전달하는 역할을 하는 패턴
- 보안이나 안정성을 높일 수 있는 패턴

MVC 패턴

- 모델, 뷰, 컨트롤 3개의 컴포넌트로 애플리케이션을 구분한 패턴

모델 컴포넌트	• 데이터 액세스
뷰 컴포넌트	• 사용자와의 상호작용
컨트롤 컴포넌트	• 전체적인 제어

- 사용자 인터페이스를 가지고 있는 많은 애플리케이션에 사용됨

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

종류

Publisher-Subscriber 패턴

- 하나의 Publisher가 다수의 Subscriber에게 상태가 변경되었음을 단방향 전파로 통지하는 패턴
- 협력 컴포넌트들의 상태를 동기화하는데 유용함
- Observer 패턴, Dependents 패턴, Event 패턴으로 사용됨

Pipes and Filters 패턴

- 데이터 스트림을 생성하고 처리하는 패턴
- 시그널 처리와 관련된 애플리케이션에 주로 사용됨

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

Layer 패턴의 장단점

Layer 패턴	
장점	단점
• 계층별 연동을 한정할 수 있어 Loosely coupled 원칙을 지킬 수 있음 • 변화에 대한 영향력을 한정할 수 있어 코딩이나 테스트를 계층별로 진행할 수 있음 • 인터페이스 정의가 잘 되어 있다면 계층을 통째로 교체할 수 있음	• 계층의 원칙을 지키기 위해 각 계층을 모두 거쳐야하므로 성능 측면에 불이익을 받을 수 있음 • 계층을 구분하기 어렵고 잘못 구분할 경우 설계 수정이 빈번히 발생할 수 있음

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

Blackboard 패턴의 장단점

Blackboard 패턴	
장점	단점
• 완벽한 해법을 찾기 어려운 경우에 사용할 수 있음 • KS(Knowledge Source), Control, Blackboard가 독립적으로 동작하여 가변성이나 유지보수성이 좋음 • KS는 타 문제 도메인에 재사용될 수 있음	• 완벽한 해법을 제시하지 못하므로 얼마동안 동작해야 하는지 알 수가 없음(성능문제) • 계산 결과가 항상 동일하지 않아 테스트가 어려움 • 많은 시간에 걸쳐 수정되어야 하므로 개발에 많은 노력이 필요

소프트웨어 아키텍처 설계

소프트웨어 아키텍처 패턴

Broker 패턴의 장단점

Broker 패턴	
장점	단점
• 컴포넌트 간의 위치 투명성을 제공 • 플랫폼 간의 Portability 제공함 • 다른 서버 시스템의 연동을 용이하게 함 • 재사용 컴포넌트 확보에 용이	• 성능에 대한 불이익 • 장애 대처율이 떨어짐 • 테스트 디버깅의 복잡함 (서버, 클라이언트 연동 시)

학습정리

1. 컴포넌트 기반 개발의 개요

- 컴포넌트 : 특정한 기능을 수행하기 위해 독립적으로 개발되고 잘 정의된 인터페이스를 가지며 다른 부품과 조립되어 애플리케이션을 구축하기 위해 사용되는 소프트웨어 부품
- 컴포넌트 기반 개발 : 소프트웨어 개발 주기의 모든 단계에서 컴포넌트를 소프트웨어 부품처럼 미리 만들어서, 애플리케이션을 조립하듯이 만들어나가는 개발 방법론

2. 컴포넌트 기반 설계

- 컴포넌트 식별은 유스케이스(Use Case)와 클래스(Class) 상관분석을 통하여 식별함

3. 소프트웨어 아키텍처 설계

- 소프트웨어 아키텍처 : 소프트웨어를 구성하는 컴포넌트와 이들 간의 상호작용 및 관계, 이들이 구성하는 소프트웨어의 설계 및 진화를 위한 원칙들의 집합
- Layer 패턴 : 가장 일반적으로 사용하는 아키텍처 패턴으로서 Subtask들을 그룹으로 묶어 사용 허가 관계를 표시하는 패턴
- Blackboard 패턴 : Shared Data, Database와 같은 데이터 중심 패턴 중에 하나이며, 명확히 정의된 문제 해법이 없을 때 문제를 풀어가는 하나의 방식을 정의한 패턴