

Documenting Software Architectures

Software Engineering Lab

김영기 책임

resious@gmail.com





Contents

1

A Collection of Software Architecture Style

C1

Module Views

C2

A Tour of Some Module Style

C3

Component-and-Connector Views

C4

Tour of Some Component-and-Connector Styles

C5

Allocation Views and a Tour of Some Allocation Styles

2

Beyond Structure: Completing the Documentation

C6

Beyond the Basics

C7

Documenting Software Interfaces

C8

Documenting Behavior

3

Building the Architecture Documentation

C9

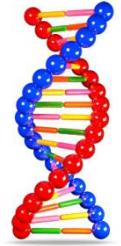
Choosing the Views

C10

Building the Documentation Package

C11

Reviewing an Architecture Document



Part I

A COLLECTION OF SOFTWARE ARCHITECTURE STYLE

I.1 Three Categories of Style

아키텍처 스타일은 아키텍처 설계에서 반복해서 나타나는 문제를 해결하고 아키텍처가 만족시켜야 하는 시스템 품질속성을 달성할 수 있는 방법을 문서로 정리한 것이다

❖ 아키텍처 스타일(Architecture Style)

- 3가지 아키텍처 스타일
 - 문서화 시 각 스타일에 해당하는 View가 적어도 하나씩 포함되어야 한다.

| | |
|--|--|
| 모듈 스타일 (Module Style) | <ul style="list-style-type: none">▪ 특화된 모듈 타입들을 소개▪ 모듈 타입들의 요소들이 어떻게 결합하는지에 대한 규칙을 지정 |
| 컴포넌트와 커넥터 스타일 (Component-and-Connector Style) | <ul style="list-style-type: none">▪ 컴포넌트와 커넥터 관점에서 실행 시 행위를 기술▪ 데이터와 컨트롤 플로우 기술 |
| 할당 스타일 (Allocation Style) | <ul style="list-style-type: none">▪ 개발환경 또는 실행 환경의 요소와 소프트웨어 요소간의 맵핑 기술 |

- 아키텍처 스타일과 아키텍처 패턴
 - 아키텍처 스타일과 아키텍처 패턴을 구별하지 않고 사용
 - 아키텍처 패턴 – 시스템의 많은 아키텍처 컴포넌트 중 일부에만 국한
 - 아키텍처 스타일 – 아키텍처 패턴보다는 좀 더 광범위한 디자인 솔루션



I.2 Style Guides

❖ 스타일 가이드 (Architecture Guide)

- 스타일을 설명하기 위한 기준 구성
 - 각각의 스타일에 대한 비교와 선택에 대한 일관적인 기술이 중요
- 스타일 가이드의 전체적인 구성과 섹션의 내용



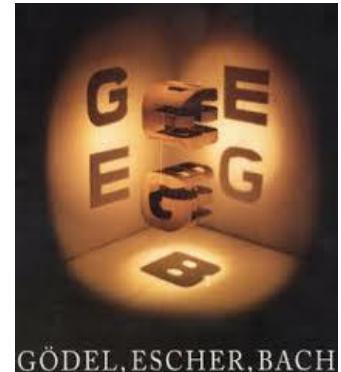
| | | |
|--|--------------------------|---|
| 개요(Overview) | | <ul style="list-style-type: none">▪ 왜 해당 스타일이 유용한지 설명<ul style="list-style-type: none">- 시스템과 해당 스타일이 시스템을 어떻게 지원하는지 기술 |
| 요소 (Element Types) 관계 (Relation Types) 특성 (Properties) | Elements | <ul style="list-style-type: none">▪ 아키텍트의 구성 요소▪ 요소 탑입을 정의하고, 스타일을 사용하는 아키텍처가 요소의 인스턴스를 어떻게 사용하는지 정의 |
| | Relations | <ul style="list-style-type: none">▪ 관계 탑입을 정의▪ 요소들간 동작 방법 및 협력 관계를 제공 |
| | Constraints | <ul style="list-style-type: none">▪ 스타일의 유효한 인스턴스를 형성하기 위한 요소(element)와 관계(relation)간의 규칙을 나열 |
| | What it's for | <ul style="list-style-type: none">▪ 스타일에서 뷰(View)에 의해 지원되는 추론의 종류를 기술 |
| | Notations | <ul style="list-style-type: none">▪ 스타일의 뷰를 문서화 할 때 이용가능하고 유용한 그래픽/텍스트 표현의 기술 |
| | Relation to other styles | <ul style="list-style-type: none">▪ 해당 스타일에서 유도된 다른 스타일에서 유도된 뷰와 연관되어야 하는지를 기술 |
| | Examples | <ul style="list-style-type: none">▪ 주어진 스타일에서 유도된 뷰의 문서화 예제를 제공 |

I.3 Choosing Properties to Document

❖ 최종 목표는 선택된 스타일에 기반한 뷰(Views)를 기술하는 것

- 뷰를 문서화하는 것은 문서화 할 요소의 특성을 결정하는 것
 - 요소의 이름(Name), 역할(Role), 책임(Responsibility)
 - 아키텍처 기반의 분석을 지원하는 특성들
 - 성능(Performance) : 요소의 최고/최하 응답시간, 요소의 최대 이벤트 처리량, ...
 - 보안(Security) : 암호화 레벨, 인증 규칙 등 ...
 - 하나의 뷰는 하나 이상의 스타일을 나타낼 수 있다.

| | | What | How | Where | Who | When | Why | | |
|---|---|-------------|-----|-------|-----|------|-----|--|-------------|
| Row 1 – Scope External Requirements and Drivers Business Function Modeling | 1 | Contextual | | | | | | | Contextual |
| Row 2 – Enterprise Model Business Process Models | 2 | Conceptual | | | | | | | Conceptual |
| Row 3 – System Model Logical Models Requirements Definition | 3 | Logical | | | | | | | Logical |
| Row 4 – Technology Model Physical Models Solution Definition and Development | 4 | Physical | | | | | | | Physical |
| Row 5 – As Built As Built Deployment | 5 | As Built | | | | | | | As Built |
| Row 6 – Functioning Enterprise Functioning Enterprise Evaluation | 6 | Functioning | | | | | | | Functioning |
| | | What | How | Where | Who | When | Why | | |



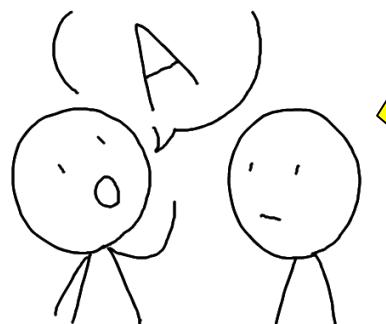
Zachman
Framework for
Enterprise Architecture

I.4 Notations for Architecture Views

❖ 표기법(Notations)

- 뷰를 문서화하기 위한 표기법은 형식성(Formality)의 정도에 따라 구별

| | |
|------------------------------------|--|
| 비공식적인 표기법 (Informal Notations) | <ul style="list-style-type: none">선택된 시스템을 시각적인 관례를 일반적인 목적의 다이어그램과 편집 도구를 이용한 표기자연어로 기술되고, 특성화 된 의미(Semantic)은 형식적으로 분석하기 어려움 |
| 준정형적 표기법 (Semiformal notations) | <ul style="list-style-type: none">미리 제공된 그래픽 요소와 구성 규칙에 따른 표준화 된 표기법에 의해 기술된 뷰요소들의 완전한 의미적 처리를 제공하지는 못한다. - UML초보적인 분석이 적용 가능 |
| 정형적 표기법 (Formal notations) | <ul style="list-style-type: none">일반적으로 수학적 방법에 기반한 정확한 의미를 갖는 표기법구문과 의미에 대한 형식 분석이 가능 - 때때로 특정 스타일에 특화됨ADLs : Architecture Description Language - 도구와 연계되어 자동화된 분석이 가능함 |



There is no greater impediment to the advancement of knowledge than the ambiguity of words
- Thomas Reid,
Scotish Philosopher





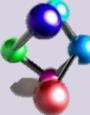
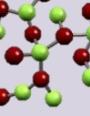
Chapter. 1

MODULE VIEWS

1.1 Overview

❖ 이번 장에서는 모듈 뷰 측면에서 아래 사항을 살펴본다.

- 요소(Elements), 관계(Relations), 그리고 특성(Properties)
- 목적(Purpose)과 표기법(Notation)
- 다른 뷰와의 관계(Relation to other views)

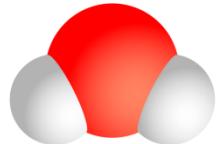
| | |
|--|---|
| 요소 (Elements)  | <ul style="list-style-type: none">▪ 모듈은 소프트웨어의 구현 단위▪ 모듈은 책임들의 응집된 셋을 제공 |
| 관계 (Relations)  | <ul style="list-style-type: none">▪ Is part of : 서브 모듈과 파트, 전체와 연관 모듈의 부분과 전체의 관계를 정의▪ Depends on : 두 모듈 사이의 종속성을 정의▪ Is a : 둘 이상의 특정 모듈 사이의 일반화/특화 관계를 정의 |
| 제약 사항 (Constraints)  | <ul style="list-style-type: none">▪ 여러 모듈 뷰들은 토플러지에 따른 제약사항(Topological Constraints)을 내포할 수 있음 |
| 목적 (What It's For)  | <ul style="list-style-type: none">▪ 코드 작성을 위한 청사진 제공▪ 영향도 분석 향상▪ 점진적 개발 계획▪ 요구사항 추적성 분석 지원▪ 코드 베이스의 구조와 시스템의 기능성 설명▪ 작업 할당, 구현 일정, 예산 정보 정의 지원▪ 지속되어야 하는 정보 구조 설명 |



1.2 Elements, Relations, Properties of Module Views

❖ 요소(Element)

- 모듈(Module)은 책임의 응집된 셋을 제공하는 S/W의 구현 단위
- 책임(Responsibility)는 아키텍처에 기여하기 위해 예상되는 것
 - 시스템 품질 속성이나 기능을 위해 수행되어야 하는 행위, 유지되는 지식, 수행하는 역할 등을 포함
- 모듈은 합성(aggregated)이나 분해(decomposed)가 가능



❖ 관계(Relation)

| Is part of | Depends on | Is a |
|-----------------|-----------------|-----------------|
| Aggregation | Association | Inheritance |



1.2 Elements, Relations, Properties of Module Views

❖ 특성 (Properties)

- 뷰에 대한 보조 문서의 부분으로 기록

| | |
|---|---|
| 이름 (Name) | <ul style="list-style-type: none">▪ 모듈을 식별하는 가장 기본적인 장치▪ 이름을 통하여 시스템 내의 역할을 짐작할 수 있는 경우가 많다. |
| 책임 (Properties) | <ul style="list-style-type: none">▪ 시스템 전반에 걸쳐 해당 모듈이 하는 역할을 알 수 있다.▪ 모듈의 책임을 기술 할 때는 모듈의 역할을 명확히 알 수 있도록 자세히 기술 |
| 인터페이스 가시성 (Visibility of Interfaces) | <ul style="list-style-type: none">▪ 모듈의 인터페이스 문서는 시스템 내에서 해당 모듈이 맡은 역할을 상세하게 정의▪ 모듈을 어떻게 호출하면 시스템 내에서 맡은 역할을 수행하게 되는지를 명세 |
| 구현 정보 (Implementation information) | <ul style="list-style-type: none">▪ 모듈은 구현의 단위이기 때문에, 아래의 정보를 기록해 두면 유용<ul style="list-style-type: none">✓ 코드 단위들 간의 맵핑✓ 테스트 정보✓ 관리 정보✓ 구현 제약 사항 |

- 위의 속성 외에도 뷰타입의 스타일별로도 속성이 더 있다.



1.3 What Module Views Are For

❖ 모듈 뷰의 사용

- 구축(Construction)

- 소스코드 작성의 청사진 :

- 모듈은 소스코드나 디렉터리 같은 물리적 구조와 대응

- 분석(Analysis)

- 요구사항에 대한 추적 용이성(Requirement Traceability)

- 모듈의 할당 책임을 가지고 시스템의 기능 요구사항 지원을 예측

- 영향도 분석(Impact Analysis)

- 시스템 변경으로 발생할 수 있는 영향을 예측하는데 활동

- 의사소통(Communication)

- 시스템이 익숙하지 않은 사람에게 시스템의 기능을 설명

- 시스템에 부여된 책임을 하향식으로 표현하는 것이 가능

- 다양한 규모의 학습 과정을 이끄는 지침으로 활용 가능



호출 내용의 문서화

설계 완결성과 모듈의
무결성이 중요

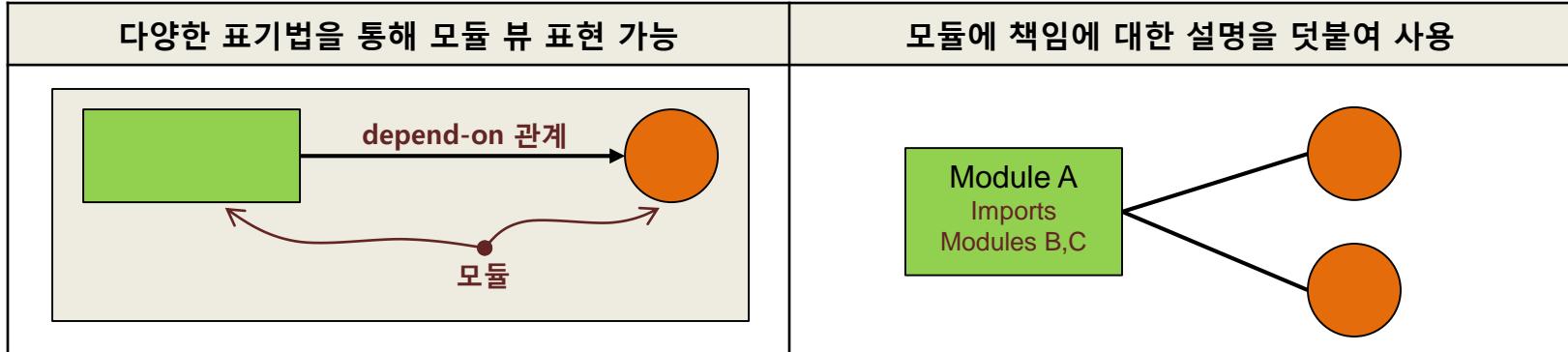


모듈 뷰타입은 S/W 기능을 나누는 것에 쓰이기 때문에 실행시간의 행위에 대한 추론은 어렵다.



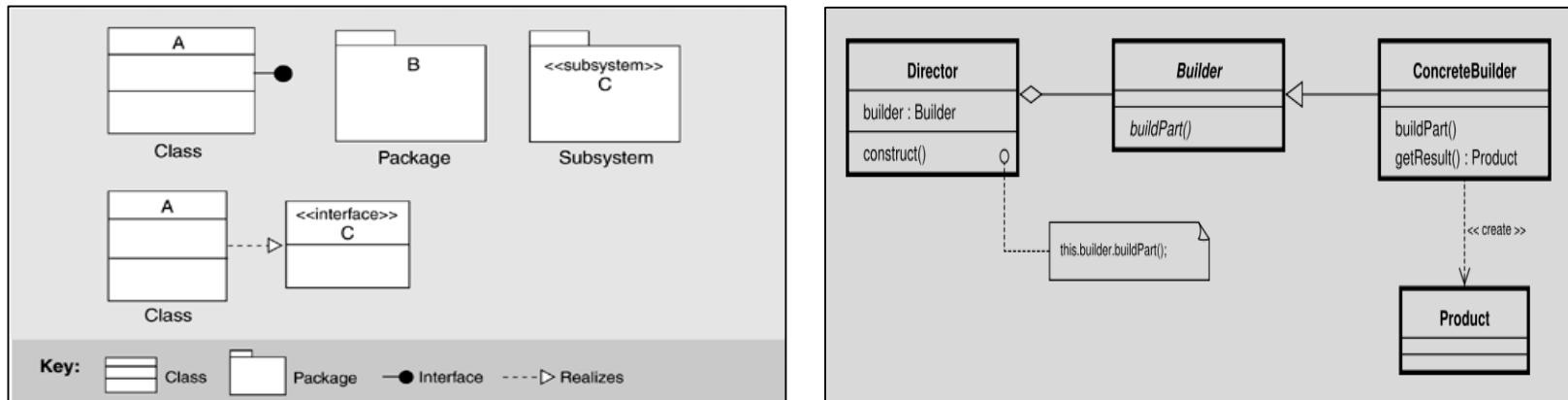
1.4 Notations for Module Views

❖ 비공식적인 표기



❖ 준-정형적 표기법(Unified Modeling Language, UML)

- 다양한 종류의 모듈을 표현하는 데 활용 가능함





1.4 Notations for Module Views

❖ DSM(Dependency Structure Matrix)

- 시스템 내의 많은 요소들과 요소간의 관계를 나타냄
 - 요소간의 영향도를 쉽게 파악할 수 있음

| | task 1 | task 2 | task 3 | task 4 | task 5 | task 6 |
|--------|--------|--------|--------|--------|--------|--------|
| task 1 | X | | | X | | |
| task 2 | | X | X | | | |
| task 3 | X | | X | | | |
| task 4 | | | | X | | |
| task 5 | | X | | | X | |
| task 6 | | | | | X | |

Design Structure Matrix

| | task 1 | task 2 | task 3 | task 4 | task 5 | task 6 |
|----------|--------|--------|--------|--------|--------|--------|
| person 1 | X | | | X | | |
| person 2 | | X | | | | |
| person 3 | | | X | | X | |
| person 4 | | | | X | | |

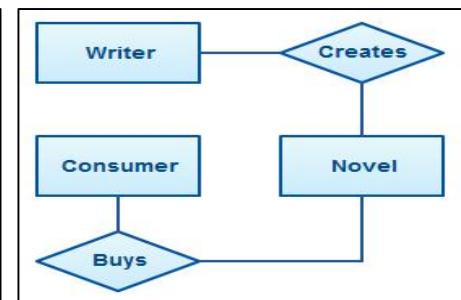
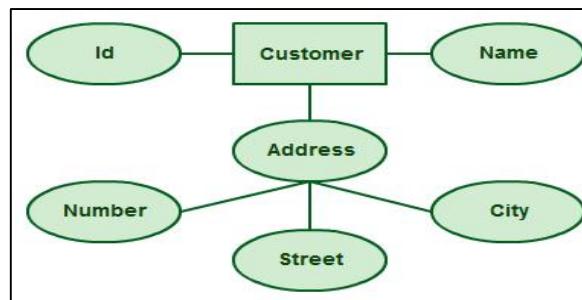
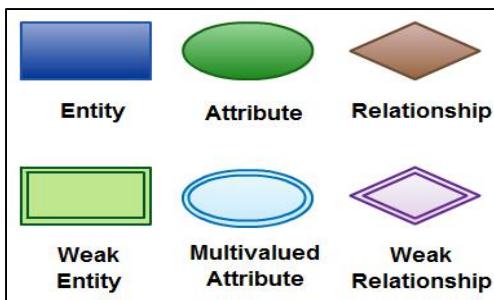
Domain Mapping Matrix

| | task 1 | task 2 | task 3 | task 4 | task 5 | task 6 | person 1 | person 2 | person 3 | person 4 |
|----------|--------|--------|--------|--------|--------|--------|----------|----------|----------|----------|
| task 1 | X | | | X | | | | | | |
| task 2 | | X | X | | X | | | | | |
| task 3 | | | X | | | X | | | | |
| task 4 | | | | X | | | | | | |
| task 5 | | | | | X | | | | | |
| task 6 | | | | | | X | | | | |
| person 1 | X | | | X | | | X | X | X | X |
| person 2 | | X | | | | | X | | X | |
| person 3 | | | X | | | | | X | | |
| person 4 | | | | X | | | | | X | |

Multiple-Domain Matrix

❖ ER 다이어그램 (Entity-Relation Diagram)

- 데이터 모델링에 특화되어 사용

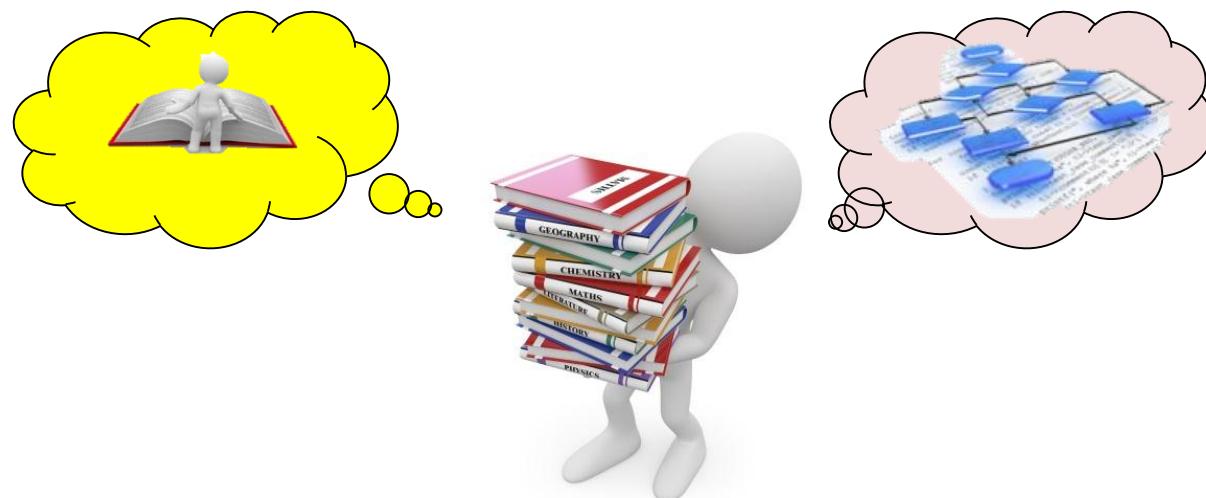




1.5 Relations to Other Views

❖ 모듈 뷰는 ...

- 일반적으로 C&C 뷰타입의 뷰들과 대응
 - 모듈 뷰의 구현 단위 = 실행시간의 수행 컴포넌트
- 다른 뷰에 필요한 정보가 과도하게 포함되는 문제
 - 제대로 알고 있지 않는 경우, 혼동을 유발시킴





Chapter. 2

A TOUR OF SOME MODULE STYLE



In this Chapter ...

❖ 6가지 주요 모듈 스타일



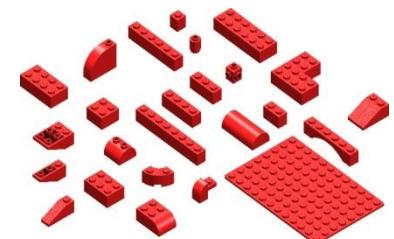
| | |
|---|--|
| 분할 스타일 (Decomposition Style) | <ul style="list-style-type: none">▪ 모듈과 서브 모듈의 구조를 보여주기 위해 사용 |
| 사용 스타일 (Uses Style) | <ul style="list-style-type: none">▪ 모듈간의 기능 종속 관계를 표시하기 위해 사용 |
| 일반화 스타일 (Generalization Style) | <ul style="list-style-type: none">▪ 모듈간의 특화 관계를 표시하기 위해 사용 |
| 계층 스타일 (Layered Style) | <ul style="list-style-type: none">▪ 계층이라 불리는 모듈의 그룹 사이의 제한된 형식에서, 사용 허용(allowed-to-use) 관계의 제한된 방식을 설명 |
| 관점 스타일 (Aspects Style) | <ul style="list-style-type: none">▪ 횡단 관심(crosscutting concerns)에 책임이 있는 관점(Aspects)이라 불리는 특정 모듈을 기술하기 위해 사용<ul style="list-style-type: none">* 핵심 관심 : 시스템이 추구하는 핵심 기능 및 가치* 횡단 관심 : 핵심 관심에 공통적으로 적용되는 공통 모듈 |
| 데이터 모델 스타일 (Data Model Style) | <ul style="list-style-type: none">▪ 데이터 엔티티 간의 관계를 보여주기 위해 사용 |



2.1 Decomposition Style

❖ 개요

- 분할 뷰는 모듈과 서브 모듈로 써의 코드의 구조(organization)을 설명
 - Is-part-of 관계에 중점
 - 시스템의 책임이 어떻게 모듈에 걸쳐 나뉘어지는가를 보여줌
 - 분할 정복 기법 (Divide-and-Conquer Technique)
 - 모듈을 더 작은 모듈로 분해하는 것은 아래 사항을 포함
 - 특정 품질 속성의 성취
 - 구현(Build) 또는 구매(Buy)에 대한 결정
 - 제품 라인 구현
 - 팀 할당
 - 모듈의 크기는 버리고 다시 시작할 수 있는 정도의 작은 크기가 좋다.
- 분할 뷰에서 정의된 모듈은 다른 뷰에서 계속해서 나타날 수 있다.
 - Uses, Layered, Generalization, and ...



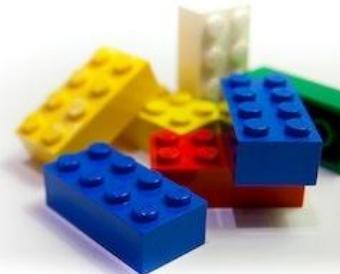


2.1 Decomposition Style

❖ 요소, 관계, 속성

▪ 분할 스타일의 요소는 모듈

- 일부 모듈은 다른 모듈과 집합하여 서브시스템으로 불린다.
- 하위 모듈은 오직 집합 모듈 내에서만 보여질 수 있다.



| | | |
|-----------------------|--|---|
| 개요 (Overview) | | <ul style="list-style-type: none">▪ 시스템을 구현 단위로 분해하는 데 사용▪ 코드의 모듈 구조와 시스템의 책임이 어떻게 분해되는가를 설명 |
| 요소 (Elements) | | <ul style="list-style-type: none">▪ 모듈 |
| 관계 (Relations) | | <ul style="list-style-type: none">▪ 분할 관계 (Decomposition) : is-part-of 의 형식을 갖는다. |
| 제약사항 (Constraints) | | <ul style="list-style-type: none">▪ 분할 그래프에서 루프는 허용되지 않는다.▪ 모듈은 오직 하나의 부모를 갖는다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">▪ 신입과 요약된 크기로 소프트웨어의 구조에 대한 왜 이유와 의사소통을 위해▪ 작업 할당에 대한 입력을 제공하기 위해▪ 지역적인 변화에 대한 사유 |



2.1 Decomposition Style

❖ 분할 스타일의 용도

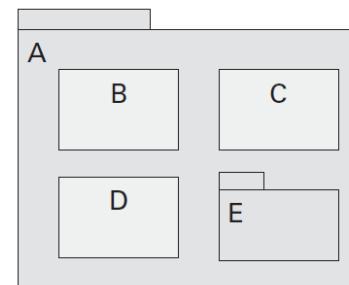
- 시스템의 책임을 상세하고, 정제 가능한 수준의 표현
 - 아키텍트의 설계 작업을 지원
 - 시스템의 전체적인 기능을 관련자에게 알려주는 도구로 활용
- 작업 할당 스타일에서 활용
 - 소프트웨어 요소를 개발 조직 단위와 매핑
- 제한적 수준의 영향도 분석 지원 가능



❖ 표기법

| Hardware Hiding Module | Behavior Hiding Module |
|-------------------------------|--------------------------------------|
| Extended Computer Module | Function Driver Module |
| Data Module | Air Data Computer Module |
| Input/Output Module | Audible Signal Module |
| Computer State Module | Computer Fail Signal Module |
| Parallelism Control Module | Doppler Radar Module |
| Program Module | Flight Information Display Module |
| Virtual Memory Module | Forward Looking Radar Module |
| Interrupt Handler Module | Head-Up Display Module |
| Timer Module | Inertial Measurement Set Module |
| Device Interface Module | Panel Module |
| Air Data Computer Module | Projected Map Display Set Module |
| Angle of Attack Sensor Module | Shipboard Inertial Nav System Module |
| Audible Signal Device Module | Visual Indicator Module |
| Computer Fail Device Module | Weapon Release Module |

[비정형적 표현]



[UML]

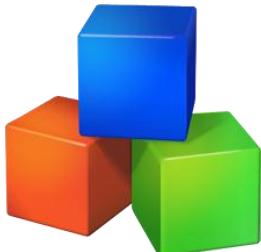
모듈은 중첩(nested)될 수 있다.



2.1 Decomposition Style

❖ 다른 스타일과의 관계

- 분할 뷰는 하나 이상의 C&C 뷰와 매핑이 가능
 - 소프트웨어의 구현 구조가 실행 구조와 어떤 연관이 있는지를 보여준다.
 - 하나의 모듈 \leftrightarrow 여러 개의 C&C 컴포넌트
 - 여러 개의 모듈 \leftrightarrow 하나의 C&C 컴포넌트
- 분할 뷰는 배치 스타일의 할당 뷰와 연관
 - 모듈과 구현의 책임을 보여준다.





2.2 Uses Style

❖ 개요

- 모듈 사용 관점에서 모듈 뷰의 Depends-on 관계가 특화된 형태
 - Uses : 하나의 모듈이 다른 모듈을 사용
 - 모듈 스타일의 기능 단위의 구성에 모듈의 사용 관계까지 보여줌
- 시스템의 모듈이 정상적으로 동작하기 위해서는 다른 모듈 필요함을 알려줌
 - 시스템의 증분적(incremental) 개발, 확장, 디버깅, 테스팅 등에 사용

❖ 요소, 관계, 속성

| | | | |
|-----------------------|--|---|-----------------------|
| 개요 (Overview) | | <ul style="list-style-type: none"> ▪ 모듈 간의 종속성이 어떤지 보여줌 | HomoLUDENS |
| 요소 (Elements) | | <ul style="list-style-type: none"> ▪ 모듈 | |
| 관계 (Relations) | | <ul style="list-style-type: none"> ▪ 사용 관계 (Uses) : Depends-on 관계의 한 형식 <ul style="list-style-type: none"> - 모듈 A가 자신의 요구를 만족시키기 위해 모듈 B의 기능에 의존 | |
| 제약사항 (Constraints) | | <ul style="list-style-type: none"> ▪ 구성상의 제약 사항 없음 <ul style="list-style-type: none"> - Loop, broad fan-out, dependency chain은 아키텍처의 기능을 떨어뜨림 | |
| 목적 (What It's For) | | <ul style="list-style-type: none"> ▪ 증분 개발과 하위 세트(Subset)의 개발 계획 ▪ 디버깅과 테스트 ▪ 변화의 효과를 측정 | |



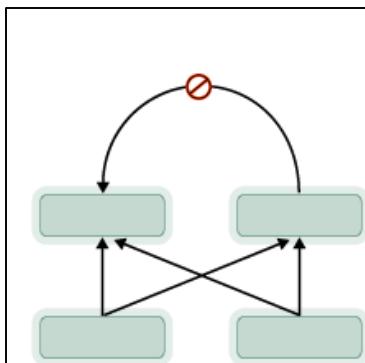
2.2 Uses Style

❖ 사용 스타일의 용도

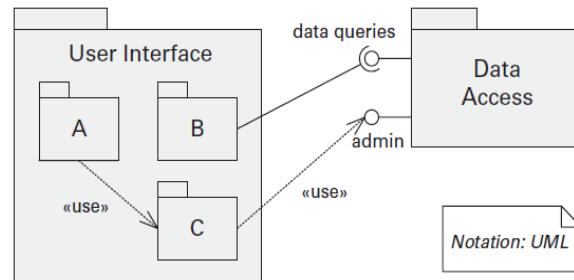
- 증분 개발과 하위 세트(Subset)의 개발 계획
- 디버깅과 테스트
- 변화의 효과를 측정
- 시스템 구축, 유지 보수 시 종속성 관리에 활용 (유지보수성 관리)



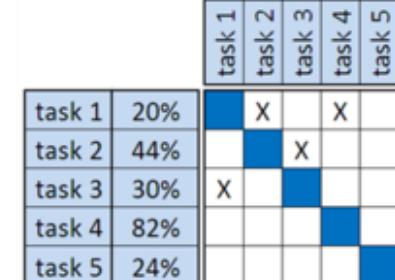
❖ 표기법



[비정형적 표현]



[UML]



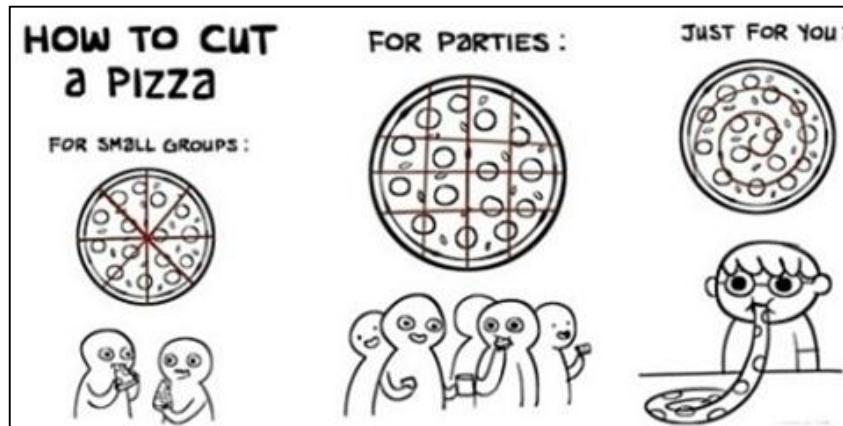
[DSM]



2.2 Uses Style

❖ 다른 스타일과의 관계

- 계층(Layered) 스타일과 관계
 - Allowed-to-use 관계
 - 개발자에게 잘 정제되지 않았지만(Coarse-grained), 자유도의 정도를 지시
 - 모듈의 일관성
 - 분할(Decomposition)은 하위 모듈에 대한 전체 모듈의 사용 관계를 포함
- 인터페이스를 명시적으로 보여줄 수 있다.





2.3 Generalization Style

❖ 개요

- 일반화를 위해 모듈 뷰의 is-a 관계가 특화된 형태
 - 일반화는 인터페이스와 구현의 상속을 나타낸다.
- 아키텍처와 개별 요소의 확장과 진화를 지원
 - 모듈의 확장은 자식모듈의 추가, 삭제, 변경을 통하여 이루어 질 수 있다.
- 모듈의 공통부분과 가변부분을 분리하기 이해 사용

부모(Parent) 모듈은
자식(Child) 모듈보다 더
일반적이다.

❖ 요소, 관계, 속성

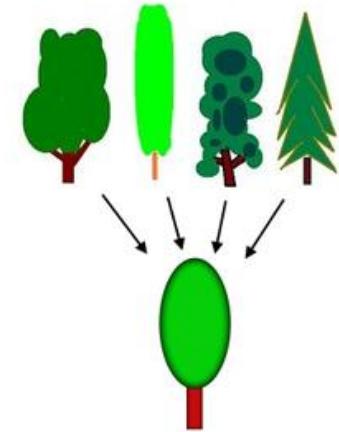
| | | |
|-----------------------|--|--|
| 개요 (Overview) | | <ul style="list-style-type: none">▪ 아키텍처와 개별요소의 확장과 진화를 지원하기 위해 is-a 관계를 사용 |
| 요소 (Elements) | | <ul style="list-style-type: none">▪ 모듈 : 모듈 뷰에서 정의된 속성 외에 "abstract" 속성이 추가됨 |
| 관계 (Relations) | | <ul style="list-style-type: none">▪ 일반화 관계(Generalization) : is-a 관계로 부터 정련됨, 특화된 상태를 나타냄 |
| 제약사항 (Constraints) | | <ul style="list-style-type: none">▪ 하나의 모듈은 여러 개의 부모를 가질 수 있다.▪ 일반화 관계에서 순환(Cycle)은 허용되지 않는다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">▪ 객체지향 설계에서 상속을 표현▪ 부분적인 진화와 확장을 설명▪ 자식의 다양한 변화와 공통성을 추출▪ 재사용을 지원 |



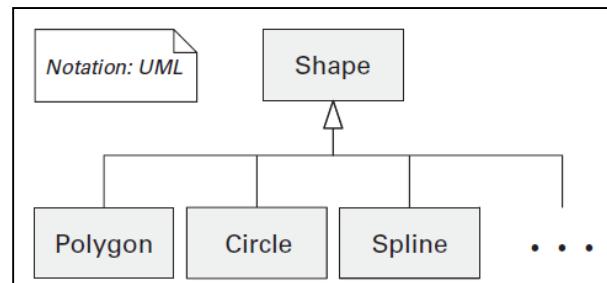
2.3 Generalization Style

❖ 일반화 스타일의 용도

- 객체지향 설계(Object-oriented Design)
 - 시스템의 객체 지향 설계에서 상속을 표현
- 확장 (Extension)
 - 한 모듈이 다른 모듈과 어떻게 다른지 쉽게 이해를 돋는다.
- 지역적인 변경과 변형 (Local change or Variation)
 - 안전한 전역적 구조와 지역적인 변경과 변형을 수용
 - 상위 모듈의 공통성과 하위 모듈의 변형 사항을 정의
- 재사용 (Reuse)
 - 추상 모듈의 정의는 재사용의 위한 기회를 창출



❖ 표기법



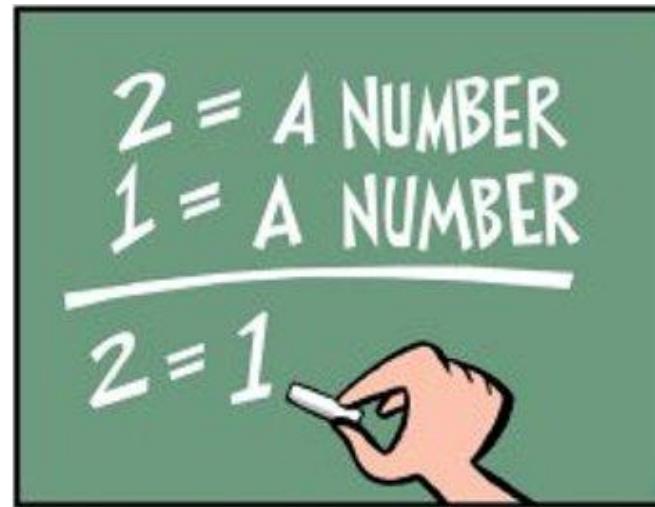
[UML]



2.3 Generalization Style

❖ 다른 스타일과의 관계

- 상속 및 인터페이스 구현 관계를 다른 모듈과의 관계로 보완
- 복잡한 모듈의 계층 구조를 포함하고 있다면 ...
 - 상속 관계를 분리된 다른 타입의 관계로 보여주는 것이 좋다.

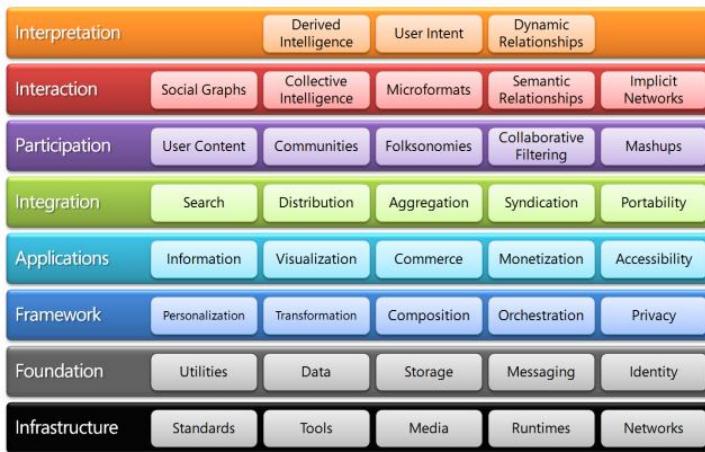




2.4 Layered Style

❖ 개요

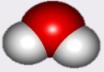
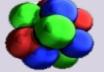
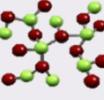
- 계층 스타일은 소프트웨어를 단위 별로 분할 하는 것을 의미
 - 제약 사항 : 계층간의 Allowed-to-use 관계가 있다.
 - 계층 구조는 변경용이성(modifiability)과 이식성(portability)을 향상시킨다.
 - 계층은 엄격한 순차적 관계에 의하여 상호작용을 한다.
 - 일반적으로 상위 계층이 하위 계층을 사용하는 관계
 - Layer bridging : 상위 계층이 바로 아래의 하위 계층이 아닌 하위 계층을 사용하는 경우
- 계층은 소스 코드를 기반으로 나누어지지 않는다.





2.4 Layered Style

❖ 요소, 관계, 속성

| | | |
|-----------------------|---|---|
| 개요 (Overview) |  | <ul style="list-style-type: none">계층간의 단방향적인 사용허용 관계를 나타낸다.계층은 응집된 서비스의 세트를 제공하는 모듈의 그룹을 말한다. |
| 요소 (Elements) |  | <ul style="list-style-type: none">계층(Layer) : 계층은 포함하고 있는 모듈을 정의해야 한다. |
| 관계 (Relations) |  | <ul style="list-style-type: none">사용 허용(Allowed-to-use) : 일반적인 종속 관계(Depends-on)의 특화디자인 시 계층간 사용 규칙을 정의해야 한다. |
| 제약사항 (Constraints) |  | <ul style="list-style-type: none">소프트웨어의 모든 부분은 정확히 한 계층에 할당되어야 한다.적어도 2개 이상의 계층이 존재해야 한다.사용 허용 관계가 순환(Cycle) 되면 않된다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none">변경용이성과 이식성 증대코드 구조 복잡성을 관리하고 개발자들에게 코드 구조에 대한 의사 소통을 촉진시킨다.재사용의 증진관심의 분리 |

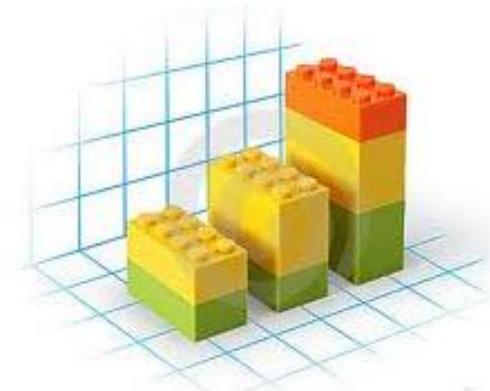




2.4 Layered Style

❖ 계층 스타일의 용도

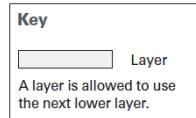
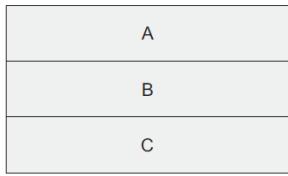
- 변경용이성과 이식성 증대
 - 코드 구조 복잡성을 관리
 - 계층은 시스템 구축 시 아키텍처의 청사진의 일부분
- 개발자들에게 코드 구조에 대한 의사 소통을 촉진시킨다.
 - 일반적인 오해 – 계층은 추가적인 실행 시 오버헤드를 가진다.
 - 구현 책임을 특정 팀에 할당하는 것을 돋는다.
- 재사용의 증진
 - 아키텍처 구조 분석을 돋는다.
- 관심의 분리
 - 변경에 대한 영향도 분석을 돋는다.
 - 변경의 범위를 정하는 것을 가능하게 한다.



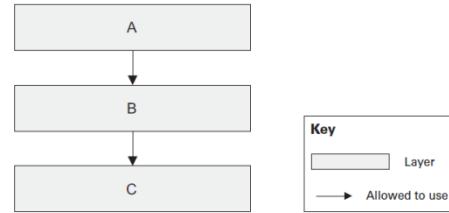


2.4 Layered Style

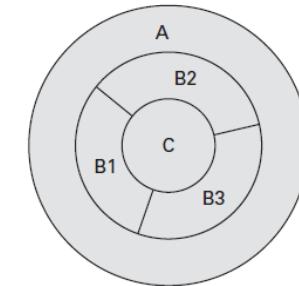
❖ 표기법



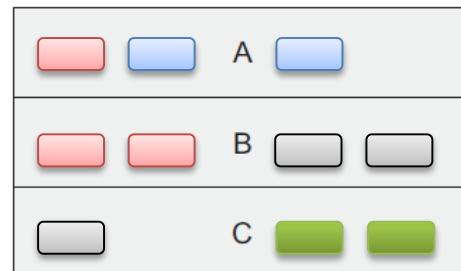
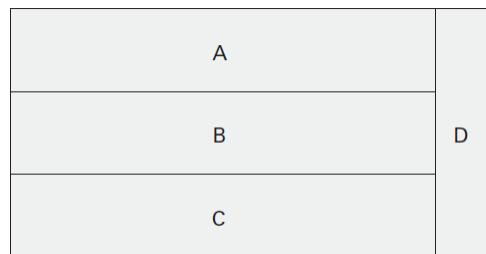
[Stack]



[Segment Layers]

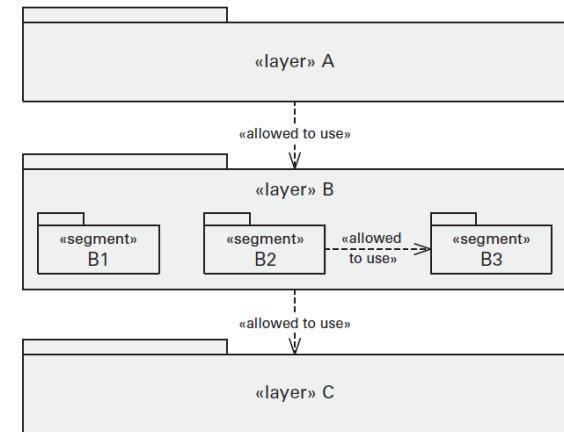


[Rings]



[Layers with a Sidebar]

[Size and Color]



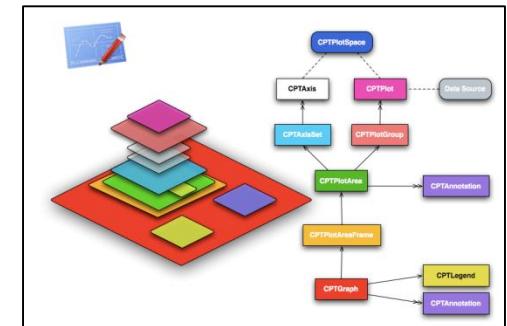
[UML]



2.4 Layered Style

❖ 다른 스타일과의 관계

- 모듈 분할 (Module Decomposition)
 - 계층 뷰의 계층은 분할 뷰의 모듈과 항상 연관되지만, 대부분 일대일 관계는 아니다.
 - 계층은 하나 이상의 모듈로 구성된다.
 - 같은 모듈의 서로 다른 서브 모듈을 다른 계층에 존재할 수 있다.
- 계층 (Tiers)
 - 계층(Layer)는 다중 계층 구조의 계층(Tier)와 혼동된다. ($\text{Layer} \neq \text{Tier}$)
 - 계층 스타일(Layered Style)은 구현 단위의 모임을 나타내며, 모듈 스타일의 한 종류
 - 다중 계층 스타일은 C&C 스타일로 실행 시의 컴포넌트에 집중
- 모듈 “사용” 스타일 (Module “uses” style)
 - 계층은 사용 허용(Allowed-to-use) 관계를 표현



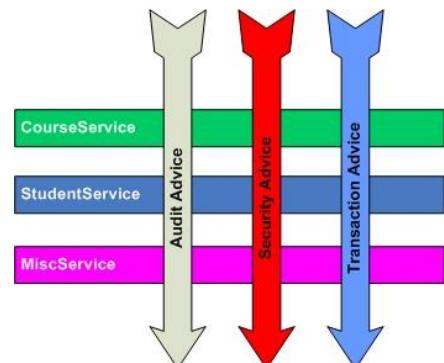


2.5 Aspects Style

❖ 개요

- 다수의 영역들에 걸쳐진 공통의 관심사로 모듈로부터 독립시킴
 - 비기능적 요소는 여러 기능들에 공통적으로 요구되는 경우가 많다.
- 측면 스타일은 크로스 커팅된 기능의 책임이 있는 모듈
 - 하나 또는 그 이상의 측면 뷰에 배치해야 된다는 것을 규정한다.
- 측면 스타일은 AOP를 사용하여 구현 시 유용

*Crosscutting concerns*은
여러 기능들에 공통적으로
요구되는 것을 의미한다.



[AOP Examples]

관점 지향 프로그래밍

(Aspect Oriented Programming, AOP)

- 핵심 관심사(core concerns)에 대한 관점과 획단 관심사(cross-cutting concerns)에 대한 관점들로 프로그램을 분해해 객체지향에서 추구하는 모듈화를 더 잘 지원하기 위한 프로그래밍 기법
- 관점지향 프로그래밍이라고 하는 것은 관심의 분리(Separation of Concerns)를 통해 기존 클래스의 비즈니스 핵심로직은 유지하고 변경은 최소화하면서 공통기능의 중복코드를 특수한 기법으로 관통하여 사용할 수 있도록 하는 것이다.



2.5 Aspects Style

❖ 요소, 관계, 속성

| | | |
|-----------------------|--|--|
| 개요 (Overview) | | <ul style="list-style-type: none">횡단 관심사(crosscutting)를 구현하는 관점 모듈(aspect modules)과 관점 모듈이 시스템 내의 다른 모듈과 어떻게 바인딩 되는가를 보여준다. |
| 요소 (Elements) | | <ul style="list-style-type: none">관점 객체 (Aspect) : 횡단 관심의 구현을 포함하는 특화된 모듈 |
| 관계 (Relations) | | <ul style="list-style-type: none">횡단(Crosscuts) : 관점 모듈을 관점 측면의 횡단 로직에 의해 영향을 받을 수 있는 다른 모듈과 엮는다. |
| 제약사항 (Constraints) | | <ul style="list-style-type: none">하나의 관점(aspect)은 관점 모듈은 물론, 하나 또는 그 이상의 정상 모듈과 횡단 (crosscut)이 가능하다.하나의 관점(aspect)는 구현에 의존하여 횡단으로 인한 무한 재귀를 유발할 수 있다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">객체 지향 설계에서 횡단 관심사(crosscutting concerns)를 모델링변경 용이성의 향상 |

❖ 관점 스타일의 용도

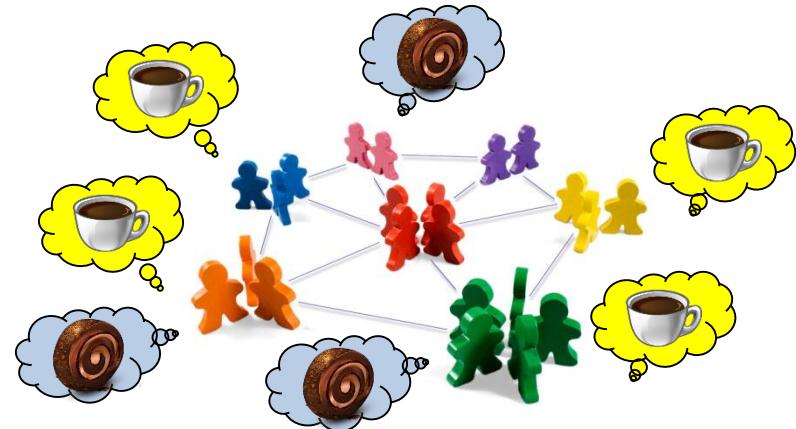
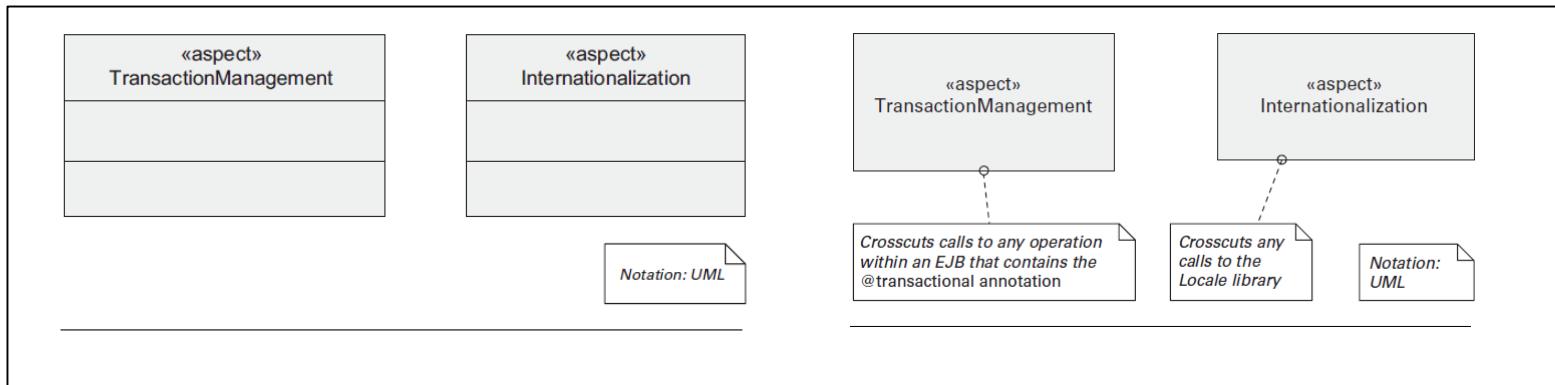
- 횡단 관심사에 대한 구현 모델링
- 변경용이성의 증진 : 기능의 꼬임 상태를 회피



2.5 Aspects Style

❖ 표기법

- 관점 객체(aspects)에 대한 UML 상의 내장된 기호는 없다.
- Crosscut 관계는 종속성의 스테레오 타입으로 표현 가능



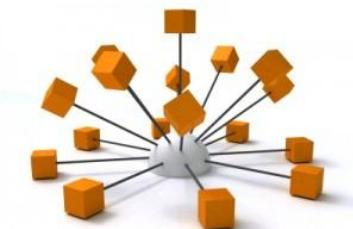


2.6 Data Model

❖ 개요

- 데이터 모델링 : 정보 시스템의 개발 프로세스의 중요한 액티비티
- 데이터 모델
 - 데이터 엔티티와 엔티티 간의 관계로 정적 정보 구조를 표현
 - ERD(Entity-Relation Diagram)으로 표현 가능

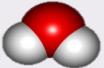
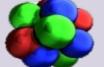
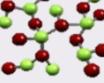
| 개념 데이터 모델 | 논리 데이터 모델 | 물리 데이터 모델 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|---|--|----|------|---------|-----|----------|---------|-----|----------------|---------|-----|---------------|---------|-----|--------------|---------|--|------------|---------------|----|------|---------|---------|--------|---------|-----|-----|---------------|--|------|----------|--|-------|---------------|----|--------|---------|--|------|-------------|--|-------------|-----------|--|-----------|---------------|--|--------|---------|
| | <p>Order</p> <pre> graph LR Order[Order] -- "has many" --> Item[Item] </pre> <p>OrderItem</p> <p>CatalogItem</p> <p>Legend: Entity + Relationship with cardinality one-to-many (crow's foot is "many")</p> | <p>PurchaseOrder</p> <table border="1"> <tr><th>PK</th><th>pkId</th><th>INTEGER</th></tr> <tr><th>FK1</th><th>clientId</th><th>INTEGER</th></tr> <tr><th>FK1</th><th>shippingInfoId</th><th>INTEGER</th></tr> <tr><th>FK2</th><th>orderDetailId</th><th>INTEGER</th></tr> <tr><th>FK4</th><th>creditCardId</th><th>INTEGER</th></tr> <tr><th></th><th>totalPrice</th><th>NUMERIC(10,2)</th></tr> </table> <p>OrderItem</p> <table border="1"> <tr><th>PK</th><th>pkId</th><th>INTEGER</th></tr> <tr><th>PK, FK1</th><th>itemId</th><th>INTEGER</th></tr> <tr><th>FK2</th><th>qty</th><th>NUMERIC(10,2)</th></tr> <tr><th></th><th>unit</th><th>CHAR(10)</th></tr> <tr><th></th><th>price</th><th>NUMERIC(10,2)</th></tr> </table> <p>CatalogItem</p> <table border="1"> <tr><th>PK</th><th>itemId</th><th>INTEGER</th></tr> <tr><th></th><th>name</th><th>VARCHAR(80)</th></tr> <tr><th></th><th>description</th><th>TEXT(400)</th></tr> <tr><th></th><th>listPrice</th><th>NUMERIC(10,2)</th></tr> <tr><th></th><th>status</th><th>INTEGER</th></tr> </table> <p>Legend: Entity + Relationship with cardinality one-to-many (crow's foot is "many")</p> <p>For each A there are 0 or more Bs, each B is related to exactly one A, As PK is needed as part of B's PK</p> <p>PK = Primary key FK# = Foreign key (bold means required column) IF = Index</p> | PK | pkId | INTEGER | FK1 | clientId | INTEGER | FK1 | shippingInfoId | INTEGER | FK2 | orderDetailId | INTEGER | FK4 | creditCardId | INTEGER | | totalPrice | NUMERIC(10,2) | PK | pkId | INTEGER | PK, FK1 | itemId | INTEGER | FK2 | qty | NUMERIC(10,2) | | unit | CHAR(10) | | price | NUMERIC(10,2) | PK | itemId | INTEGER | | name | VARCHAR(80) | | description | TEXT(400) | | listPrice | NUMERIC(10,2) | | status | INTEGER |
| PK | pkId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FK1 | clientId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FK1 | shippingInfoId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FK2 | orderDetailId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FK4 | creditCardId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | totalPrice | NUMERIC(10,2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PK | pkId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PK, FK1 | itemId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FK2 | qty | NUMERIC(10,2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | unit | CHAR(10) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | price | NUMERIC(10,2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PK | itemId | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | name | VARCHAR(80) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | description | TEXT(400) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | listPrice | NUMERIC(10,2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | status | INTEGER | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |





2.6 Data Model

❖ 요소, 관계, 속성

| | | |
|-------------------------------|---|--|
| 개요 (Overview) |  | <ul style="list-style-type: none">데이터 엔티티와 엔티티 간의 관계를 나타낸다. |
| 요소 (Elements) |  | <ul style="list-style-type: none">데이터 엔티티 (Data Entity) : 시스템에 저장하거나 표현되어야 할 필요가 있는 정보를 가지고 있는 객체<ul style="list-style-type: none">- 특성으로 이름, 데이터 속성, 기본 키(Primary Key), 엔티티에 대한 사용자 권한 규칙 등 |
| 관계 (Relations) |  | <ul style="list-style-type: none">일대일(One-to-one) , 일대다(One-to-many), 다대다(Many-to-many) 관계일반화(Generalization)/특화(Specialization)집합(Aggregation) |
| 제약사항 (Constraints) |  | <ul style="list-style-type: none">기능적인 종속 관계를 피해야 한다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none">시스템 내에서 사용되는 데이터를 기술한다.데이터 모델의 변경에 따른 영향도 분석을 수행한다. (확장성 분석)중복과 데이터의 일관성 결여를 회피함으로써 데이터 품질 향상을 강제화데이터에 접근하는 모듈의 구현에 대한 가이드 |

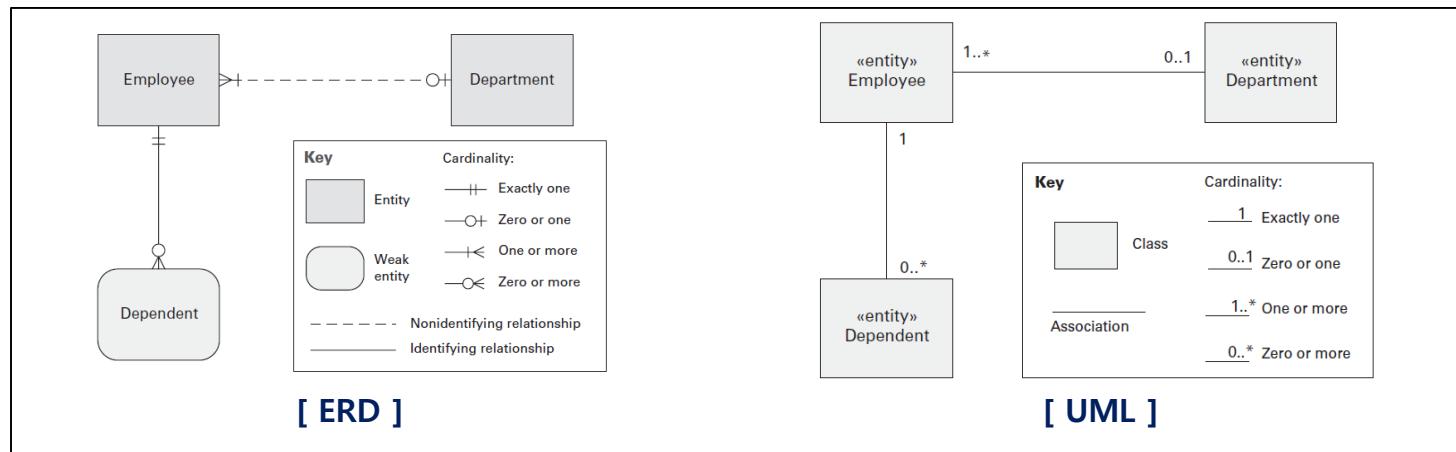


2.6 Data Model

❖ 데이터 모델의 용도

- 이해당사자와의 의사소통을 촉진
 - 도메인 분석과 요구사항 추출
 - 성능 요구사항, 비정규화(demoralization), 최적화(optimization), 디자인 결정에 활용
 - 정보 시스템 : 변경 용이성의 분석
 - 구조적 측면의 데이터 무결성에 대한 강제화를 돋는다.
 - 데이터 모델에 기반하여 물리적 데이터베이스 생성 스크립트 작성
 - 응용 프로그램 개발자에게 데이터베이스 접근 코드를 작성하는 것을 돋는다.

❖ 표기법





2.6 Data Model

❖ 다른 스타일과의 관계

- 모듈 뷰의 모듈과 본질적으로 관련이 있음
 - 모듈은 본질적으로 데이터를 포함
 - 객체지향에서 데이터 저장을 위해 관계형 데이터베이스를 사용
- 데이터 저장소는 공유 데이터 뷰에 기술
- 배치 뷰는 데이터 저장소의 하드웨어 머신으로의 할당을 표현
- 데이터 모델과 여러 데이터 저장소의 맵핑 관계의 문서화
 - 분산 데이터베이스나 데이터베이스 복제 솔루션을 사용할 때 유용





Chapter. 3

COMPONENT-AND-CONNECTOR VIEWS

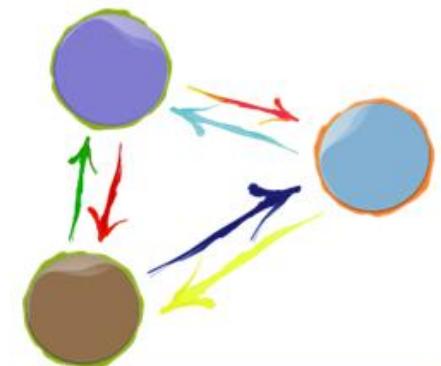




3.1 Overview

❖ 컴포넌트와 커넥터 뷰(Component & Connect View, C&C View)

- 실행시간 나타나는 요소들로 구성된 모델을 다룸
 - 선과 도형 위주의 다이어그램
 - 정형화 되지 않는 경우 불명확, 일관성 결여의 여지
 - 실행 시간에 나타나는 시스템의 전체 윤곽을 보여줌
- 컴포넌트
 - 프로세스, 객체, 클라이언트, 서버, 데이터 저장 공간
- 커넥터
 - 통신 연결, 통신 규약, 정보 흐름, 공유 저장소
 - 커넥터는 2개 이상의 대상을 연결할 수도 있다.





3.2 Elements, Relations, and Properties of C&C Views

| | | |
|------------------------|--|---|
| 요소 (Elements) | | <ul style="list-style-type: none">컴포넌트 : 기본적인 처리 단위와 데이터 저장 공간을 나타낸다.커넥터 : 컴포넌트 사이의 상호 작용 방식 |
| 관계 (Relations) | | <ul style="list-style-type: none">붙임 (Attachments) : 컴포넌트의 포트는 특정 커넥터의 역할과 관련이 있다.인터페이스 위임 (Interface Delegation) : 임의의 상황에서 컴포넌트 포트는 내부적인 하위 구조에 대한 하나 이상의 포트와 관련이 있다. |
| 제약 사항 (Constraints) | | <ul style="list-style-type: none">컴포넌트는 오직 커넥터와 연결된다. 다른 컴포넌트와는 연결될 수 없다.커넥터는 오직 컴포넌트와 연결된다. 다른 커넥터와는 연결될 수 있다.붙임은 호환 가능한 포트와 역할 사이에만 만들어 질 수 있다.인터페이스 위임은 두 개의 호환 가능한 포트 사이에서만 정의된다.커넥터는 고립되어 나타날 수 없으며, 반드시 컴포넌트와 연결되어야 한다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">시스템이 어떻게 동작하는지를 보여준다.실행 요소의 구조와 행위를 규정함으로 개발에 도움을 준다.성능, 신뢰성, 가용성과 같은 런타임 시스템 품질 속성에 대한 근거가 된다. |



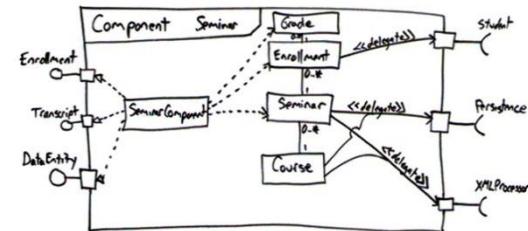
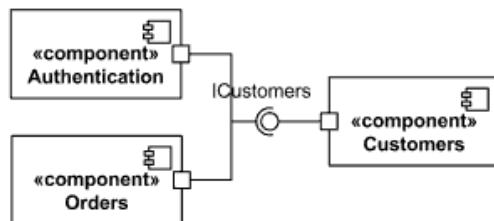
3.2 Elements, Relations, and Properties of C&C Views

❖ 요소 (Elements)

- C&C 뷰의 요소는 컴포넌트와 커넥터
 - 시스템 상의 실행 시간의 현상을 보여준다.
- C&C 뷰는 시스템의 실행 시 구성 형태를 그래프로 표현
 - 컴포넌트를 커넥터에 연결하여 관계를 나타낸다.

❖ 컴포넌트(Components)

- 컴포넌트의 이름은 컴포넌트에 부여된 기능을 가리켜야 한다.
 - 이름으로 시각적으로 표현된 요소와 보조 문서를 연계 시킬 수 있어야 한다.
- 포트(Port) : 컴포넌트가 가지는 인터페이스
 - 컴포넌트가 환경과 잠재적인 상호작용의 특정 접점을 정의한다.
 - 컴포넌트의 포트에 대한 문서화는 명확하게, 특히 포트의 개수와 종류





3.2 Elements, Relations, and Properties of C&C Views

❖ 커넥터(Connectors)

- 실행 시간에 두 개 이상의 컴포넌트 사이의 상호작용이 일어나는 경로
 - 프로시저 호출, 비동기 메시지, 이벤트 전송, 파이프 等
 - 때때로 커넥터를 통해 나타내는 상호작용은 통신규약으로 설명
- 역할(Role) : 커넥터가 가지는 인터페이스
 - 커넥터의 상호작용의 접점을 정의
 - 컴포넌트들이 커넥터를 활용해 상호작용 시 따라야 하는 방식

통신규약(Protocol)은 어떤 상호작용이 일어나면서 발생하는 이벤트나 동작의 패턴을 설명

❖ C&C 타입과 인스턴스 (Types and Instances)

- C&C 뷰 내의 컴포넌트와 커넥터는 C&C Type의 인스턴스를 기술한다.
 - C&C 뷰를 문서화 할 때
 - 어떤 아키텍트 스타일이 사용되었는지 명확히 하라
 - 뷰에서 추가적인 컴포넌트나 특화된 커넥터를 문서화하라
 - 같은 다이어그램에 여러 타입과 인스턴스를 함께 쓰는 것은 좋지 않다.

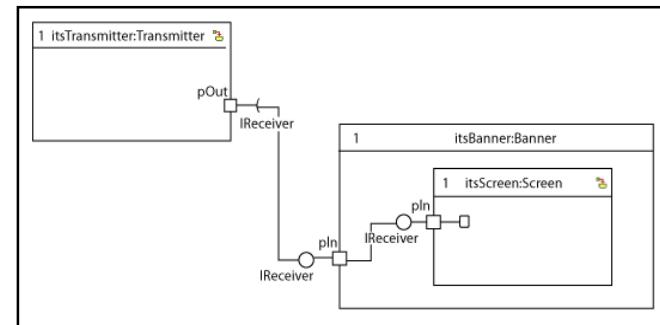
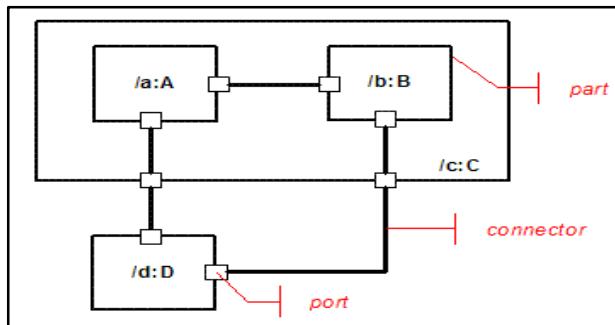
타입들은 스타일 가이드에 정의되어 있으며, 하나의 스타일에 제약 사항이 더해지면 새로운 스타일로 특화될 수 있다.



3.2 Elements, Relations, and Properties of C&C Views

❖ 관계(Relations)

- 불임(Attachment)
 - 커넥터가 어떤 컴포넌트에 연결될 것인지 정의
 - 컴포넌트의 포트와 커넥터의 역할의 연관을 나타냄
 - 스타일에 정의된 의미적인 제약사항 내에서 컴포넌트의 포트와 커넥터의 역할이 상호 호환되어야 한다.
- 인터페이스 위임(Interface Delegation)
 - 컴포넌트나 커넥터가 하위 구조를 가질 때
 - 내부적인 구조와 외부적인 인터페이스의 관계를 나타냄
 - 일부 표기법은 이러한 관계를 나타내기 위해 특정 그래픽 요소를 제공





3.2 Elements, Relations, and Properties of C&C Views

❖ 특성(Properties)

- 모든 요소는 이름(Name)과 타입(Type)을 가진다.
 - 추가적인 특성은 컴포넌트 또는 커넥터의 타입에 의존한다.
- 일반적인 속성



분석을 위한 뷰?
의사 소통을 위한 뷰?
상황에 따라 속성에 대한 선택이 필요하다. !!!

| | |
|-----------------------------------|--|
| 신뢰성 (Reliability) | <ul style="list-style-type: none">▪ 컴포넌트나 커넥터에서 주로 장애가 발생하는 곳은?▪ 이 속성은 시스템 전반적인 신뢰성을 가능하게 해준다. |
| 성능 (Performance) | <ul style="list-style-type: none">▪ 컴포넌트에 부하가 걸리 때 반응 속도의 변화는?▪ 커넥터에서 예상되는 지연시간과 처리량은?▪ 지연시간이나 처리량, 버퍼 요구량과 같은 시스템 속성을 정할 수 있게 한다. |
| 자원 요구량 (Resource requirements) | <ul style="list-style-type: none">▪ 컴포넌트와 커넥터에서 필요로 하는 처리 및 저장 용량은?▪ 제시된 하드웨어 구성이 적합한지 평가하는데 사용 |
| 기능성 (Functionality) | <ul style="list-style-type: none">▪ 요소가 수행하는 기능은 무엇인가?▪ 시스템에서 수행되는 연산 전반에 대한 근거를 추측해 볼 수 있다. |
| 보안 (Security) | <ul style="list-style-type: none">▪ 암호화나 감사 추적, 인증과 같은 시스템 보안 사항을 제공/강화하는가?▪ 시스템의 보안 취약성을 알아보는데 유용 |
| 동시성 (Concurrency) | <ul style="list-style-type: none">▪ 컴포넌트가 분리된 프로세스나 스레드에서 실행되는가?▪ 동시에 동작하는 컴포넌트의 성능을 분석/시뮬레이션하고 데드락(dead lock) 문제를 정의 |
| 계층 (Tier) | <ul style="list-style-type: none">▪ 계층 구조에서 컴포넌트가 위치하는 계층은?▪ 각 계층의 플랫폼 요구사항과 빌드와 배포 절차를 정의하는 데 도움 |



3.3 What C&C Views Are For

❖ C&C 뷰타입의 용도

- 시스템이 어떻게 동작하는지 알고자 할 때 사용
- 시스템 실행 시 품질 속성에 대한 근거를 찾을 때 사용
 - 성능, 신뢰성, 가용성
- C&C 뷰를 활용하면 다음 질문의 답을 얻을 수 있다.
 - 시스템에서 가장 기본적으로 실행되는 컴포넌트는 무엇이고, 어떤 식으로 상호작용을 하는가?
 - 공유 데이터 저장공간은 주로 어떤 것을 사용하는가?
 - 시스템의 어느 부분을 얼마나 복제해 둘 것인가?
 - 시스템이 수행되는 동안 데이터는 어떻게 훌러 다니는가?
 - 통신하는 실세 사이에 사용되는 상호작용 프로토콜은 무엇인가?
 - 시스템 내에서 병렬적으로 작동하는 부분은 어디인가?
 - 시스템이 실행하는 동안 어떤 방식으로 시스템 구조가 변경될 수 있는가?





3.4 Notations for C&C Views

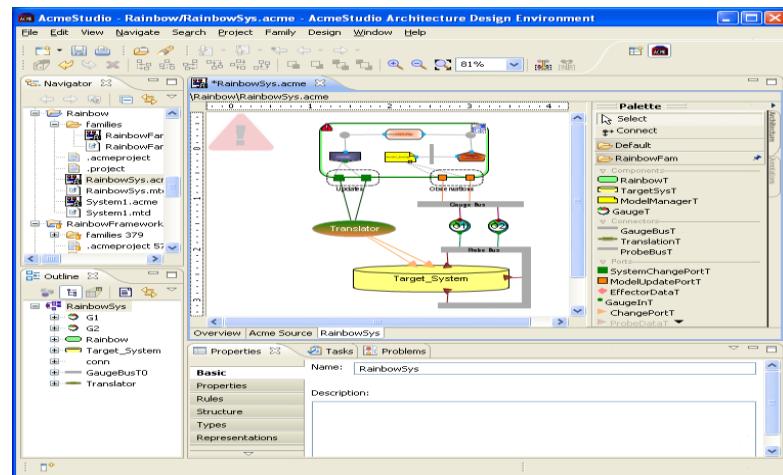
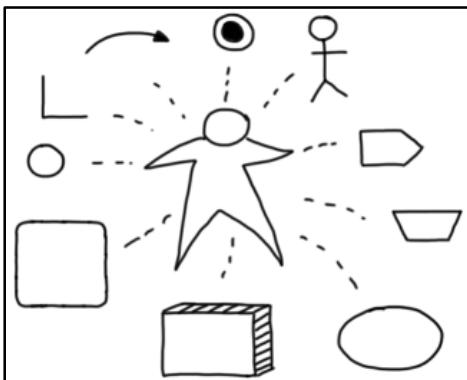
❖ 비공식적인 표기법

- Box-and-line 다이어그램

- 제한된 의미를 전달 → 사람마다 표기법 해석이 다른 문제 발생
- 엄격하고 깊이 있는 문서화를 위해서는 ...
 - 각각의 컴포넌트와 커넥터 타입을 분리된 시각화 형식으로 표현
 - 타입의 이름은 의미를 포함해야 한다.
 - 커넥터의 경우 화살표의 방향 등에 대한 의미를 명확히 해야 한다.

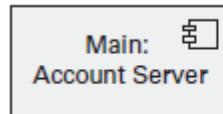
❖ 정형적 표기법

- 아키텍처 표기 언어(ADLs)

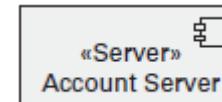


3.4 Notations for C&C Views

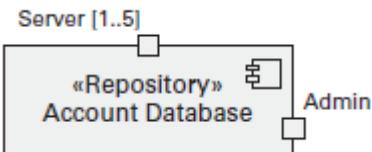
❖ 준-정형적 표기법(Unified Modeling Language, UML)



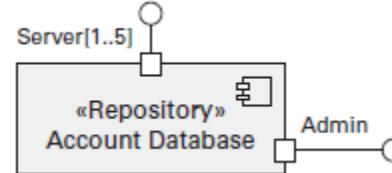
[Component]



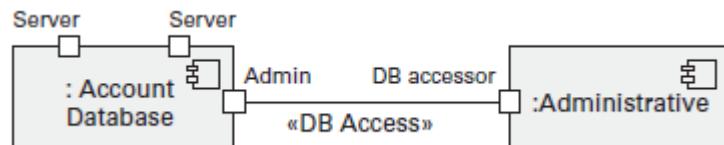
[Specialized Component]



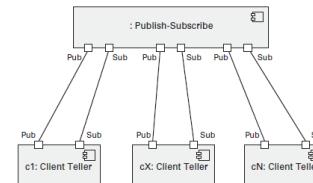
[Port]



[Port with Interface]



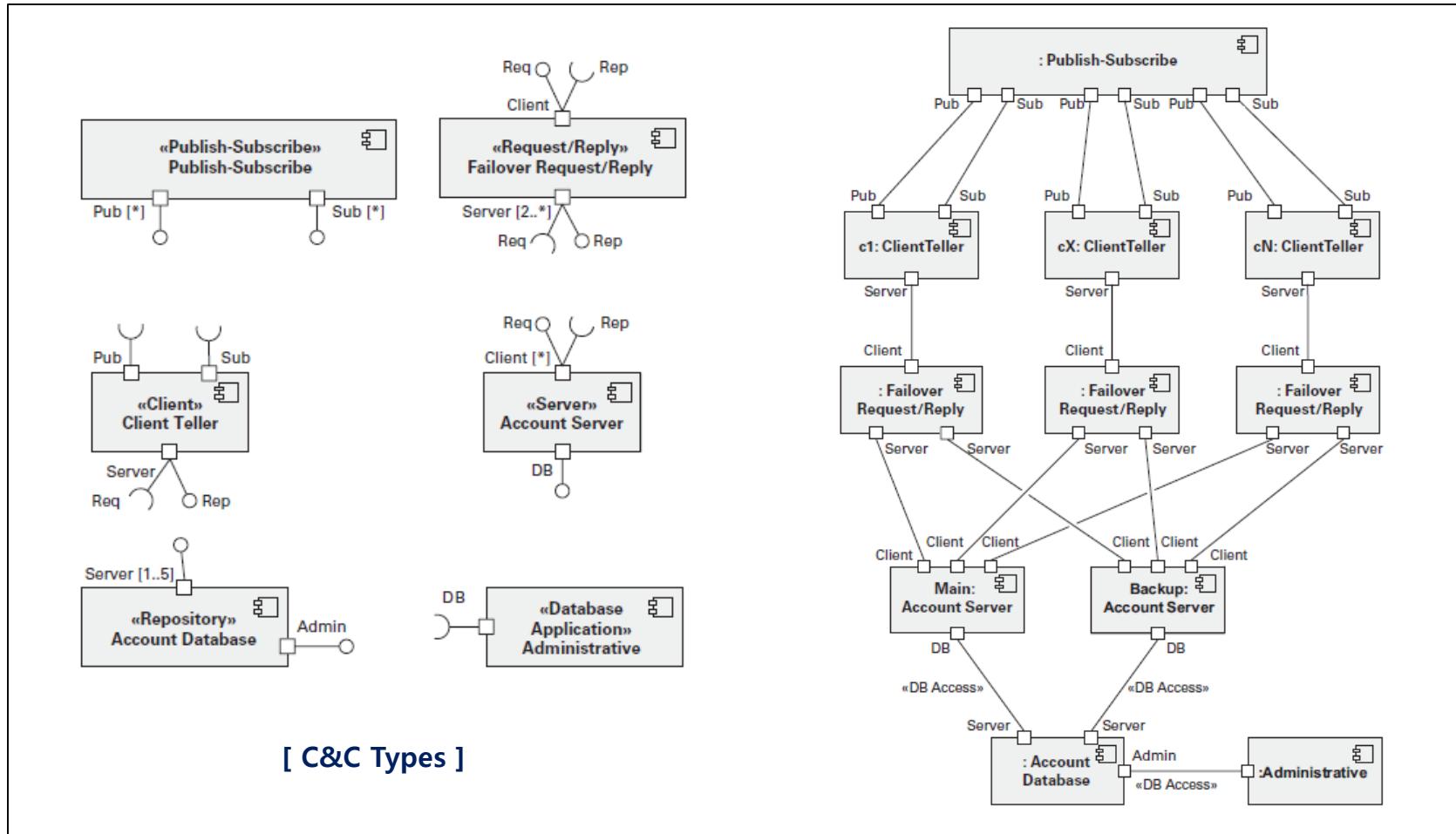
[Connector]





3.4 Notations for C&C Views

❖ 준-정형적 표기법(Unified Modeling Language, UML)

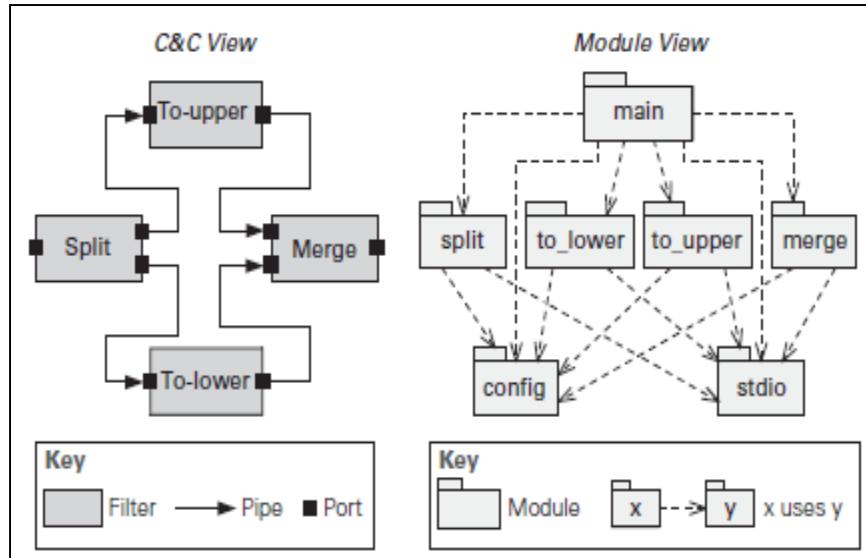




3.5 Relation to Other Kinds of Views

❖ C&C 뷰와 모듈 뷰 사이에는 요소들 간의 대응 관계

- 동일 코드 모듈이 C&C 뷰에서는 여러 요소에 의해 실행 될 수 있다.
- C&C 뷰의 한 요소는 여러 모듈의 정의된 코드를 실행 시킬 수 있다.
- C&C 컴포넌트는 주위 환경과 상호작용을 하는 접점이 여러 개 있을 수 있으며 각 지점은 동일한 모듈 인터페이스로 정의 될 수 있다.
- 모듈 뷰의 모든 요소가 C&C 뷰에 표현되지 않는다.



| C&C View | Module View |
|-------------------|-------------------------|
| System as a whole | main |
| Split | Split, config, stdio |
| To-lower | to-lower, config, stdio |
| To-upper | to-upper, config, stdio |
| Merge | merge, config, stdio |
| Each pipe | stdio |

[C&C 뷰와 모듈 뷰간의 Mapping]



Chapter. 4

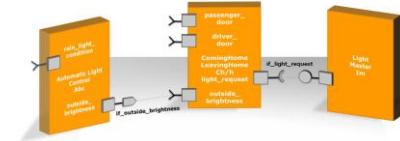
TOUR OF SOME COMPONENT-AND-CONNECTOR STYLES



4.1 An Introduction to C&C Styles

❖ C&C Views

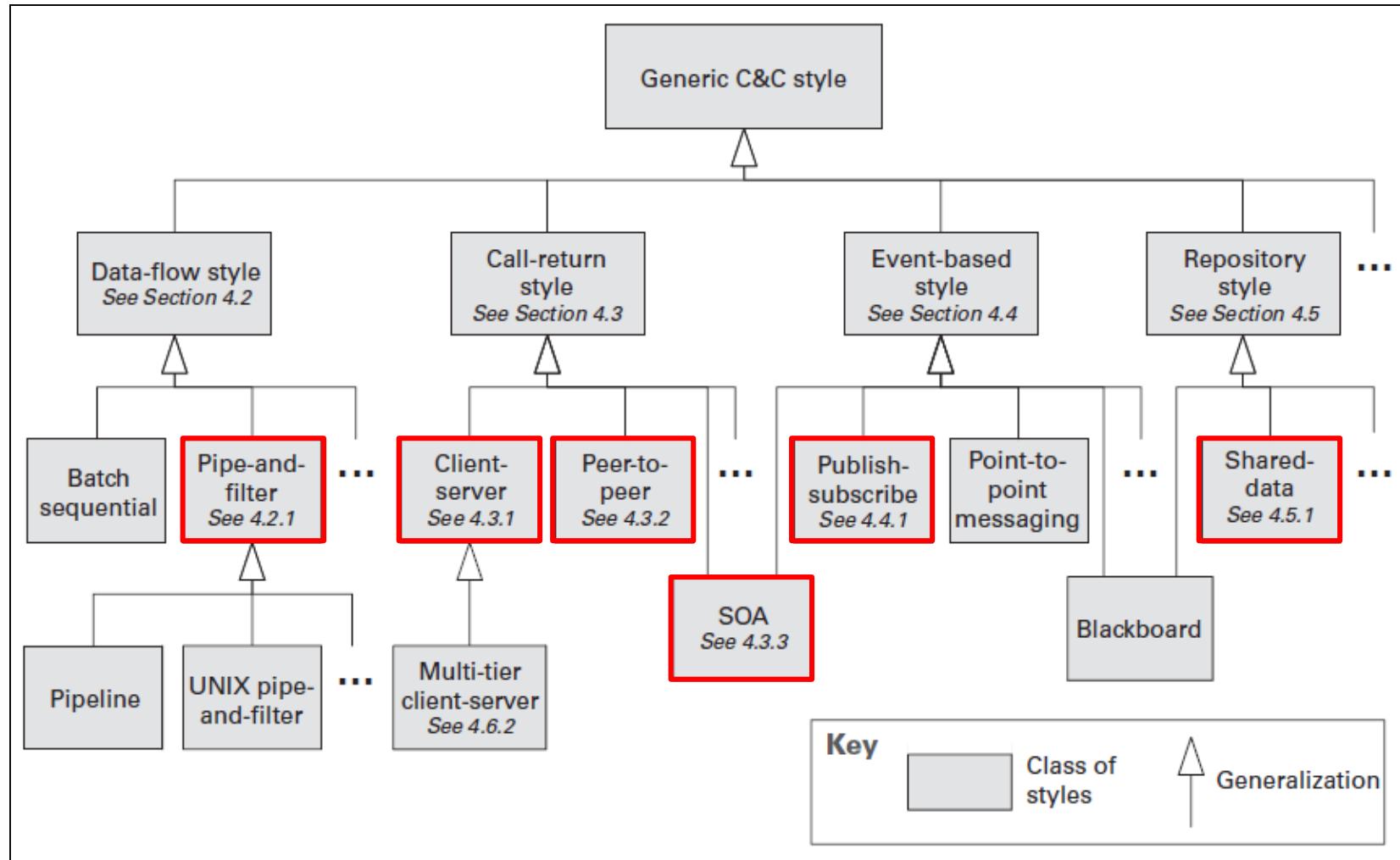
- 시스템의 실행 측면의 특성을 본다.
 - 계산 모델과 설계된 시스템에서의 데이터와 컨트롤 흐름을 기술한다.
- C&C 스타일의 선택은 시스템의 실행 시 특성에 의존한다.
 - 또한, 문서화의 사용 의도에 의존할 수 있다.



| | |
|---|--|
| 호출-반환 스타일 (Call-return Style) | <ul style="list-style-type: none">▪ 한 컴포넌트가 다른 컴포넌트가 제공하는 기능의 동기적인 호출을 통해 상호작용하는 스타일 |
| 데이터 흐름 스타일 (Data flow Style) | <ul style="list-style-type: none">▪ 시스템에 걸친 데이터의 흐름에 의해 계산이 구동되는 스타일 |
| 이벤트 기반 스타일 (Event-based Style) | <ul style="list-style-type: none">▪ 컴포넌트들이 비동기적 이벤트나 메시지에 의해 상호작용하는 스타일 |
| 저장소 스타일 (Repository Style) | <ul style="list-style-type: none">▪ 컴포넌트들이 영구적이고, 공유되는 대규모 데이터의 집합체를 통하여 상호작용 하는 스타일 |

4.1 An Introduction to C&C Styles

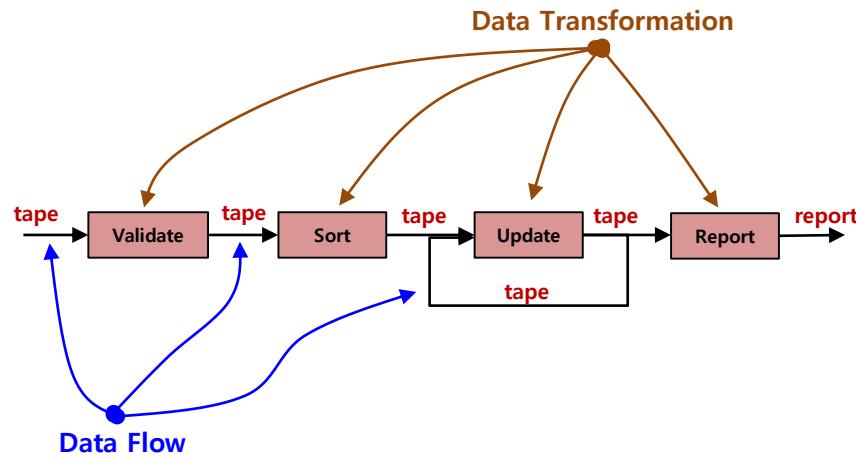
❖ A partial representation of the space of C&C style



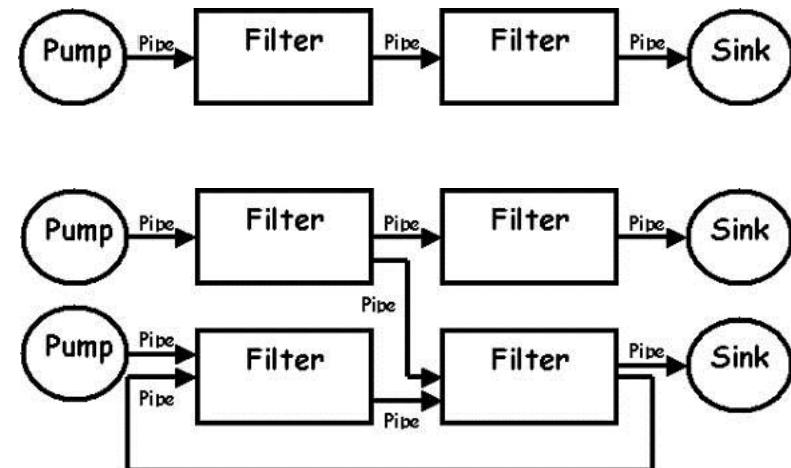
4.2 Data Flow Styles

❖ 데이터 흐름 스타일 (Data Flow Style)

- 데이터 흐름 스타일은 계산 모델을 포함한다.
 - 컴포넌트(Component) : 데이터 변환기로서 동작
 - 커넥터 (Connector) : 한 컴포넌트의 출력을 다른 컴포넌트의 입력으로 전달
- 데이터 흐름 스타일



[Batch sequential]



[Pipe-and-filter]

4.2.1 Pipe-and-Filter Style

❖ 개요

- 데이터 스트림이 연속적으로 변환
 - 필터에 도착한 데이터는 변환 후 다음 파이프를 통해 다음 필터로 이동
 - 하나의 필터는 여러 포트로 데이터를 보낼 수 있다.
 - Ex) 신호처리 시스템, 유닉스 파이프 이용 시스템

❖ 요소, 관계, 속성

| | | |
|-------------------------------|---|---|
| 요소 (Elements) |  | <ul style="list-style-type: none">▪ 필터 (Filter) : 입력 포트로 부터 들어온 데이터를 변환하여 출력 포트로 보낸다.▪ 파이프 (Pipe) : 필터의 출력을 다른 필터의 입력으로 보내는 커넥터 |
| 관계 (Relations) |  | <ul style="list-style-type: none">▪ 붙임(Attachment) : 필터의 출력 포트와 파이프의 데이터 입력 역할을 묶어주고, 필터의 입력 포트와 파이프의 출력 역할을 묶어준다. (상호작용하는 그래프를 형성) |
| 계산모델 (Computational Model) |  | <ul style="list-style-type: none">▪ 시스템의 외부 입력으로부터 외부 출력까지 데이터의 변환이 필터들에 의하여 연속적으로 수행된다. |
| 제약 사항 (Constraints) |  | <ul style="list-style-type: none">▪ 파이프는 필터의 출력 포트에서 필터의 입력 포트로 연결된다.▪ 필터는 연결된 파이프를 통해 지나는 데이터 타입과 일치해야 한다.▪ 비순환적이거나, 선형적인 시퀀스와 같이 컴포넌트의 연관을 제한 할 수 있다. (파이프라인)▪ 컴포넌트가 stdin, stdout 과 같은 특정 이름의 포트를 갖도록 규정할 수 있다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none">▪ 필터의 독립성에 기인한 재사용 향상▪ 데이터 프로세싱의 병렬화를 통한 성능 향상▪ 전체적인 행위에 대한 근거의 단순화 |



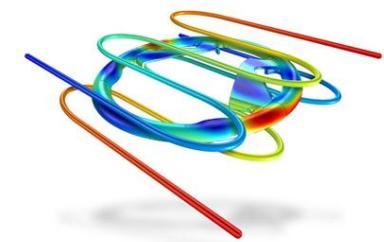
4.2.1 Pipe-and-Filter Style

❖ 파이프 앤 필터 스타일의 용도

- 일반적으로 데이터를 변환하는 시스템에서 사용
 - 전체적인 변환 작업이 독립적인 단계로 나뉘어질 수 있다.
 - 각각의 단계는 입력 데이터의 부분적 변환의 책임을 가진다.
- 파이프와 필터를 방탕으로 할 수 있는 시스템 분석
 - 집합 변환 유도, 시스템의 성능 추론 등

❖ 다른 스타일/모델과의 관계

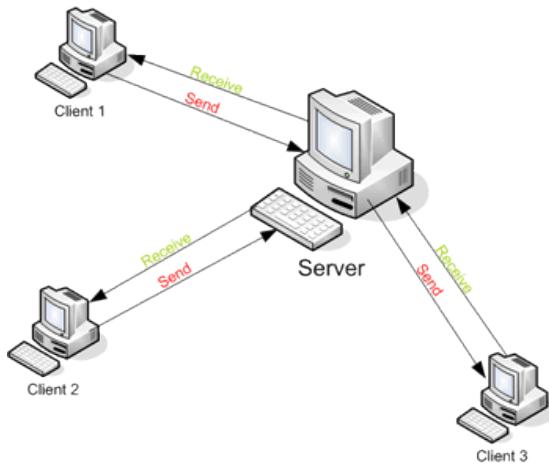
- 데이터 플로우 뷰와 다름
 - 파이프 앤 필터
 - 커넥터는 필터 사이의 선으로 표현, 연산 상의 스트림의 전송을 의미
 - 데이터 플로우
 - 연산상의 의미 없이 데이터 통신 관계를 표현



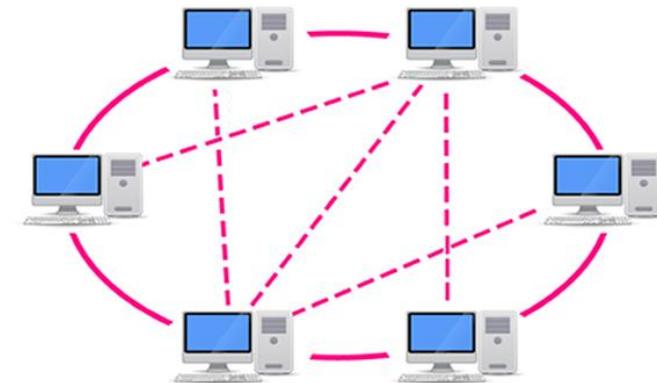
4.3 Call-Return Style

❖ 호출-반환 스타일

- 컴포넌트가 다른 컴포넌트들에 의해 호출될 수 있는 서비스 세트를 제공하는 계산 모델을 포함
 - 컴포넌트는 호출된 서비스가 완료될 때까지 서비스 호출을 멈춤
 - 커넥터는 요청 측의 서비스의 호출과 제공자의 결과의 반환에 책임



[Client-Server Style]



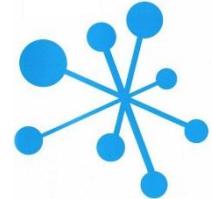
[Peer-to-peer Style]



4.3.1 Client-Server Style

❖ 개요

- 컴포넌트 들이 서비스를 서로 요청하면서 상호작용
 - 짹을 지어 통신하고, 클라이언트가 요청 시 서비스를 제공하는 측과 짹을 지음
 - 서버는 하나 이상의 인터페이스로 여러 가지 서비스를 제공
 - 클라이언트는 서버가 제공하는 서비스를 사용하지 않거나 하나 이상 사용 가능



❖ 요소, 관계, 속성

| | | |
|---------------------------------------|---|--|
| 요소 (Elements) |  | <ul style="list-style-type: none"> ▪ 클라이언트 (Client) : 다른 컴포넌트에 서비스를 요청 ▪ 서버 (Server) : 다른 컴포넌트에 서비스를 제공 ▪ 요청/응답 커넥터 : 서버에서 제공하는 서비스에 대한 클라이언트의 비대칭적 호출을 의미 |
| 관계 (Relations) |  | <ul style="list-style-type: none"> ▪ 붙임 (Attachment) : 클라이언트를 커넥터의 요청 역할에 묶어주고, 서버를 커넥터의 응답 역할에 묶어준다. 또한 어느 서비스가 어느 클라이언트로부터 요청을 받을 수 있는지도 결정한다. |
| 계산모델 (Computational Model) |  | <ul style="list-style-type: none"> ▪ 클라이언트가 상호작용을 시작하고, 필요에 따라 서버에서 제공하는 서비스를 요청하고 요청에 대한 결과를 기다린다. |
| 제약 사항 (Constraints) |  | <ul style="list-style-type: none"> ▪ 클라이언트는 요청/응답 커넥터를 통하여 서버와 연결된다. ▪ 서버는 다른 서버에 대하여 클라이언트가 될 수 있다. ▪ 특화는 다음과 같은 제한 사항을 내포 : 포트에 대한 붙임 개수, 서버 사이의 허용된 관계 ▪ 컴포넌트는 계층적으로 나눌 수 있다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none"> ▪ 변경용이성과 재사용 증대, 확장성과 가용성 향상, 종속성과 보안 및 성능 분석 |

4.3.1 Client-Server Style

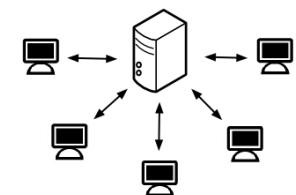
❖ 클라이언트-서버 스타일의 용도

- 공통 서비스를 분리시켜 시스템에 대한 이해도를 높임
- 기능들을 묶어 그룹으로 만듦
 - 시스템 배치나 기존 시스템에서 제공되는 서비스와 상호 운용 시 기초 정보
- 성능, 확장성이, 신뢰성이 제고
- 신뢰성, 보안, 성능 분석



❖ 다른 스타일과의 관계

- C&C 스타일들은 서비스나 데이터의 생산자와 소비자를 분리
 - C-S 스타일은 프로그래밍 언어에서 다루는 프로시저나 함수, 메소드의 호출을 일반화
 - Peer-to-Peer : C-S 스타일에서 나타나는 비대칭성이 없음
- Client와 Server는 그룹으로 묶어 분산 환경상의 여러 기계에 배치
 - 단계적 구조를 형성하는 경우가 많음





4.3.2 Peer-to-Peer Style

❖ 개요

- 컴포넌트들이 서비스를 교환하는 피어로서 서로 직접 상호작용
 - C-S 스타일에 나타나는 비대칭성이 없는 요청/응답의 상호작용
 - 커넥터는 양방향 상호작용에 필요한 복잡한 통신규약을 갖는다.

❖ 요소, 관계, 속성

| | | |
|-------------------------------|--|---|
| 요소 (Elements) | | <ul style="list-style-type: none">▪ 피어(Peer) : 컴포넌트▪ 호출-반환 (Call-Return) 커넥터 : 피어 네트워크를 연결, 다른 피어를 찾고, 서비스를 요청 |
| 관계 (Relations) | | <ul style="list-style-type: none">▪ 붙임(Attachment): 호출-반환 커넥터로 피어들이 묶여진다. |
| 계산모델 (Computational Model) | | <ul style="list-style-type: none">▪ 다른 피어에게 서비스를 요청하는 피어들이 협동하는 방식으로 동작 |
| 제약 사항 (Constraints) | | <ul style="list-style-type: none">▪ 포트나 역할마다 허용 가능한 붙임의 개수에 제한을 둘 수 있다.▪ 특정한 피어 컴포넌트는 라우팅, 인덱싱, 피어 찾기 기능을 제공할 수 있다.▪ 특화는 컴포넌트가 다른 컴포넌트를 알 수 있다는 가시성 제한을 포함할 수 있다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">▪ 가용성과 확장성 향상▪ 파일 공유, 인스턴스 메시지, 그리드 컴퓨팅과 같은 고분산 시스템을 가능하게 한다. |



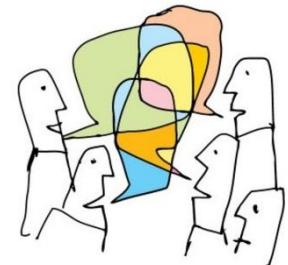
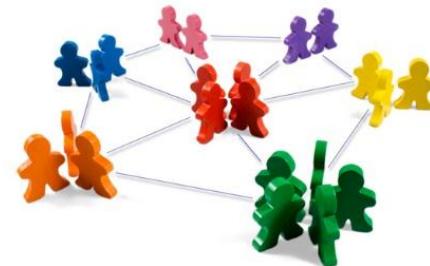
4.3.2 Peer-to-Peer Style

❖ P2P 스타일의 용도

- 분산 플랫폼 상에 시스템 배치 시 유연성을 높임
- 특정 컴포넌트의 부담을 줄임
 - 서버 용량과 기반 구조 지원에 필요한 책임을 분산 시킴
 - 데이터나 중앙 서버 저장소의 갱신 과정에서 일어나는 통신 빈도를 줄임
 - 데이터를 개별 저장하는 부담이 생김
- 분산 컴퓨팅 애플리케이션에 활용
 - 자원의 효율적 사용
 - 작업 결과의 직접 공유

❖ 다른 스타일과의 관계

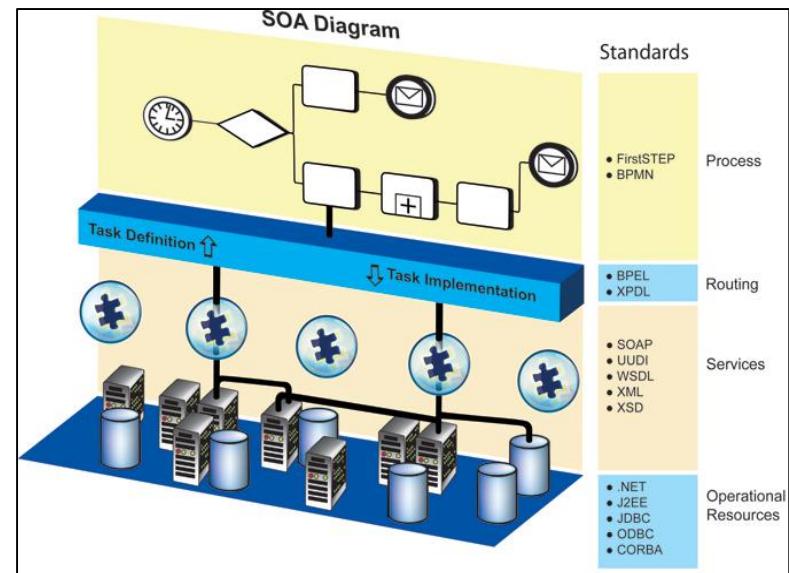
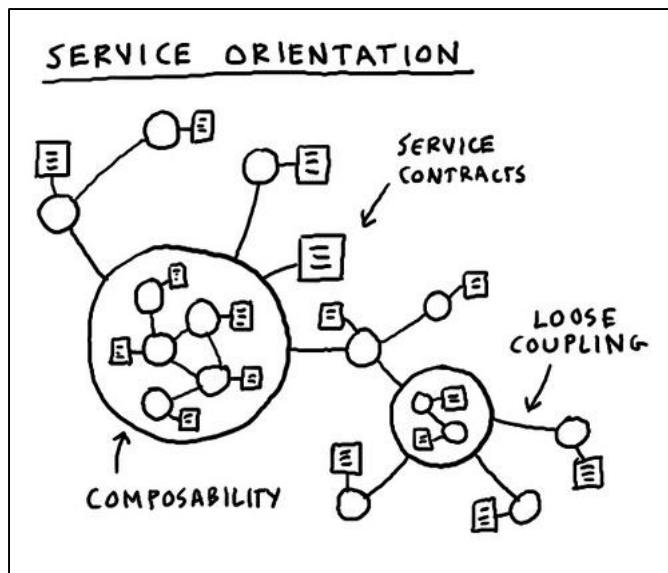
- 계층 구조가 없기 때문에 C-S 시스템보다 일반적인 형태



4.3.3 Service-Oriented Architecture Style

❖ 개요

- SOA는 서비스를 제공하거나 소비하는 분산 컴포넌트의 집합으로 구성
 - 서비스를 제공하는 컴포넌트와 소비하는 컴포넌트는 다른 구현 언어와 플랫폼을 사용하는 것이 가능
- 서비스들은 대부분 독립적이다.
 - 컴포넌트들은 독립적으로 배치
 - 컴포넌트들은 서로 다른 시스템이나 조직에 소속





4.3.3 Service-Oriented Architecture Style

❖ 요소, 관계, 속성

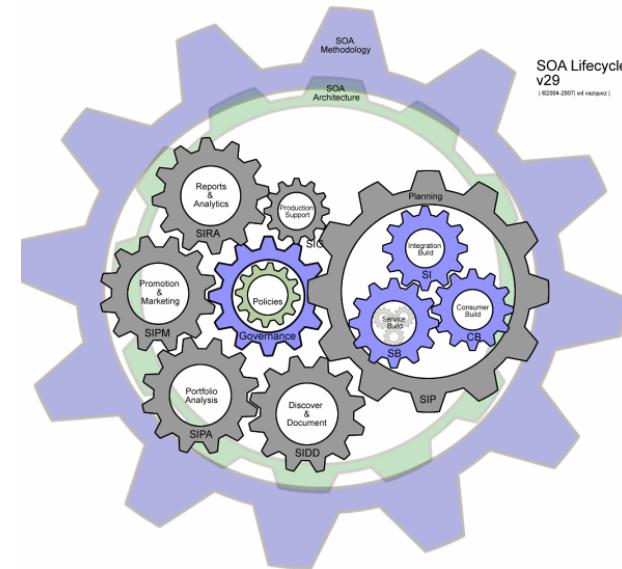
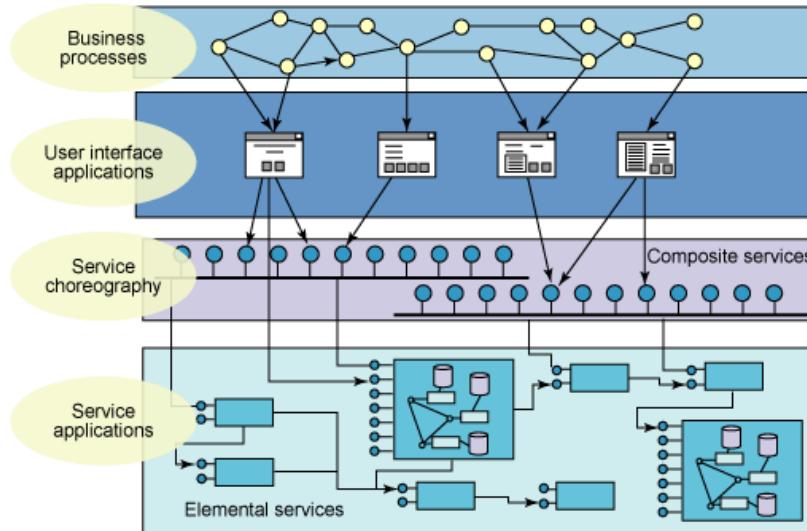


| | | |
|-------------------------------|--|---|
| 요소 (Elements) | | <ul style="list-style-type: none">서비스 제공자 (Service Providers) : 하나 이상의 서비스를 공개된 인터페이스를 통해 제공서비스 소비자 (Service Consumer) : 직접 또는 중간 매개체를 통해 서비스를 호출한다.엔터프라이즈 서비스 버스 (ESB) : 서비스 제공자와 소비자 사이의 메시지에 대한 라우팅과 전달 할 수 있는 중간 요소서비스 레지스토리 (Registry of Services) : 제공자가 서비스를 등록하기 위해 사용하며, 소비자는 실행 시 서비스를 찾기 위해 사용오케스트레이션 서버 (OrchestrationServer) : 서비스 소비자와 제공자 사이의 상호작용을 조정하고, 비즈니스 워크플로우가 정의된 스크립트에 기반한 스크립트 제공SOAP 커넥터 : 웹 서비스 사이의 동기 통신을 위해 SOAP 프로토콜을 사용메세징 커넥터 : 비동기 메시지 교환을 위해 제공되는 메시지 시스템을 사용 |
| 관계 (Relations) | | <ul style="list-style-type: none">붙임(Attachment) : 각각의 커넥터에 대해 서로 다른 종류의 포트가 가능하다. |
| 계산모델 (Computational Model) | | <ul style="list-style-type: none">네트워크를 통해 서비스를 제공하거나 소비하는 협동하는 컴포넌트의 집합으로 이루어진다.종종 워크플로우 모델의 한 종류로 기술된다. |
| 제약 사항 (Constraints) | | <ul style="list-style-type: none">서비스 소비자는 서비스 제공자와 연결되지만, 중간에 컴포넌트가 사용될 수 있다.엔터프라이즈 서비스 버스(ESB)는 hub-and-spoke 방식을 유도한다.서비스 제공자는 서비스 소비자가 될 수 있다.특정 SOA 패턴은 추가적인 제한 사항을 포함한다. |
| 목적 (What It's For) | | <ul style="list-style-type: none">서로 다른 플랫폼이나 인터넷을 통해 실행되는 분산 컴포넌트의 상호운용을 허용한다.레거시 시스템과 통합동적 구조 변경을 허용한다. |

4.3.3 Service-Oriented Architecture Style

❖ SOA 스타일의 용도

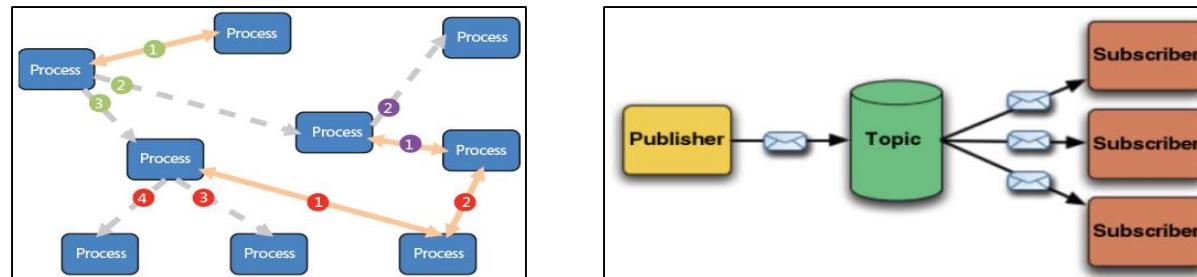
- 상호 운용성의 활용
 - 서비스 제공자와 소비자는 서로 다른 플랫폼에서 운영
 - 서로 다른 시스템이나 레거시 시스템과 통합
- 때때로 외부 서비스 요소와 상호작용하기 위해 인터넷 활용



4.4 Event-based Style

❖ 이벤트 기반 스타일

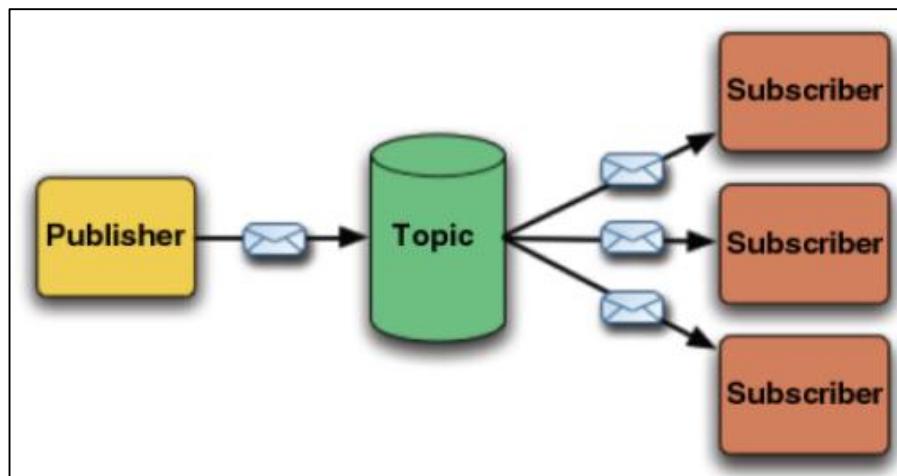
- 컴포넌트들이 비동기 메시지를 통하여 통신
 - 느슨한 결합(loosely coupled)을 통한 컴포넌트들의 제휴
 - 이벤트를 통하여 다른 컴포넌트의 행위를 트리거
- 다양한 이벤트 스타일이 존재
 - Point-to-Point
 - 호출-반환 스타일과 유사하나, 동시성이 강함
 - 이벤트 송신 축은 이벤트 수신측에서 처리하는 동안 동작을 정지하지 않음
 - Multi-party
 - 이벤트가 여러 컴포넌트에 전달
 - 발행 구독 시스템 (publish-subscribe system)



4.4.1 Publish-Subscribe Style

❖ 개요

- 이벤트의 알림을 통하여 상호작용
 - 커넥터의 기본 형태는 이벤트 버스
- 컴포넌트는 여러 가지 이벤트를 구독할 수 있음
 - 발행된 이벤트가 이벤트를 받아볼 모든 구독자에게 전달하는 책임
 - 발행 구독 실행시간 기반 구조에서 처리
- 메시지의 생산자와 소비자를 분리하고자 할 때 사용
 - 나중에 변경이 용이하다.



4.4.1 Publish-Subscribe Style

❖ 요소, 관계, 속성

| | | |
|-------------------------------|--|--|
| 요소 (Elements) |  | <ul style="list-style-type: none">모든 C&C 컴포넌트 타입 : 이벤트를 발행하거나 구독하는 인터페이스를 가진다.발행 구독 커넥터 (Publish-subscriber connector) : 컴포넌트가 발행하거나 구독하기를 원하는 이벤트를 알리거나 듣는다. |
| 관계 (Relations) |  | <ul style="list-style-type: none">붙임 (Attachment) : 어느 컴포넌트가 이벤트를 통지하고, 어떤 컴포넌트가 이벤트를 수신할지를 등록하는 것을 지시하여 컴포넌트와 발행 구독 커넥터를 묶어준다. |
| 계산모델 (Computational Model) |  | <ul style="list-style-type: none">컴포넌트가 이벤트를 구독한다. 컴포넌트가 이벤트를 발행하면, 커넥터가 모든 구독자에게 이벤트를 전달한다. |
| 제약 사항 (Constraints) |  | <ul style="list-style-type: none">모든 컴포넌트는 이벤트 배분자와 연결된다.컴포넌트는 두 타입의 포트 모두를 가짐으로써 발행자와 구독자의 역할 모두를 할 수 있다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none">알려지지 않은 수신자에게 이벤트를 보낸다 (이벤트 생산자를 소비자로 부터 고립시킨다.)GUI 프레임워크, 메일링 리스트, 공개 게시판, 소셜 네트워크의 핵심 기능을 제공 |

❖ 발행 구독 스타일의 용도

- 미지의 수신자에게 이벤트와 메시지를 보낸다.
 - 발신자의 변경 없이 수신자를 추가할 수 있다.
- 응용 프로그램에서 UI의 분리

4.4.1 Publish-Subscribe Style

❖ 다른 스타일과의 관계

- 지속성을 지원하지 않는 공유 데이터 블랙보드로 볼 수 있음
 - 블랙보드 스타일 : 데이터베이스가 이벤트 발행
 - 발행 구독 스타일 : 어떤 컴포넌트라도 이벤트 발행이 가능
- 묵시적인 호출은 호출 반환 스타일과 결합되는 경우가 많다.
 - 서비스 호출 또는 비동기적 이벤트 발행에 컴포넌트가 동기적을 상호작용 하는 경우



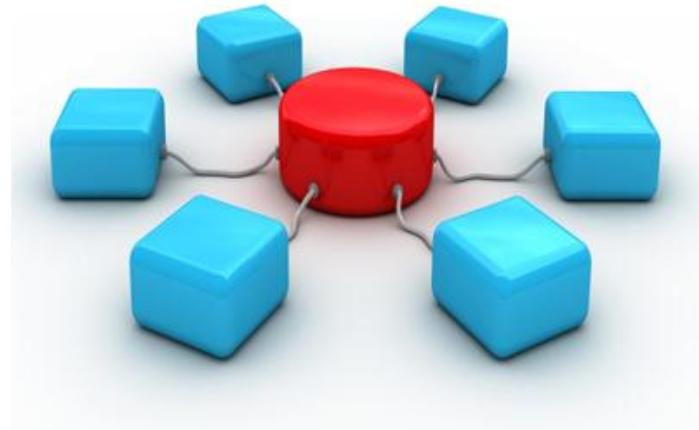


4.5 Repository Styles

❖ 저장소 스타일 (Repository Style)



- 하나 이상의 저장소라 불리는 컴포넌트를 포함한다.
 - 저장소(Repository) : 대규모의 영구 데이터의 집합을 유지하는 저장소
- 컴포넌트들이 저장소에 데이터를 쓰거나 읽는다
 - Ex) 데이터베이스 시스템
- 데이터 접근자
 - 저장소에서 말하는 공유 데이터 스타일을 따라 상호작용을 시작하는 것에 대한 책임이 있다.



4.5.1 Shared-Data Style

❖ 개요

- 영속적인 데이터를 교환하는 형태의 상호작용
- 데이터의 접근자가 여러 개 존재해야 한다
- 데이터를 영속적으로 보관할 수 있는 공유 데이터 저장소도 최소한 하나 이상 존재
- 예로는, 데이터베이스 시스템이나 지식 기반 시스템



4.5.1 Shared-Data Style

❖ 요소, 관계, 속성

| | | |
|-------------------------------|---|---|
| 요소 (Elements) |  | <ul style="list-style-type: none">저장소 컴포넌트 (Repository Component) : 데이터를 저장, 데이터 성능 관련 특성, 데이터 분산, 접근 허용수에 대한 특성을 가진다.데이터 접근자 컴포넌트 (Data accessor Component)데이터 읽기-쓰기 커넥터 (Data read and write connector) : 트랜잭션 유무의 특성 |
| 관계 (Relations) |  | <ul style="list-style-type: none">붙임 (Attachment) : 어느 데이터 접근자가 어느 데이터 저장소에 연결되는지 결정 |
| 계산모델 (Computational Model) |  | <ul style="list-style-type: none">공유 저장소에서 데이터 접근자들 사이의 통신을 중재통신은 데이터 접근자에서 먼저 시작할 수도 있고, 데이터 저장소에서 먼저 시작할 수도 있음 |
| 제약 사항 (Constraints) |  | <ul style="list-style-type: none">데이터 접근자는 데이터 저장소와 상호작용한다. |
| 목적 (What It's For) |  | <ul style="list-style-type: none">영속적인 데이터에 여러 컴포넌트가 접근하는 것을 허용데이터 생산자를 데이터 소비자로부터 분리하여 변경용이성 향상을 제공 |

❖ 공유 데이터 스타일의 용도

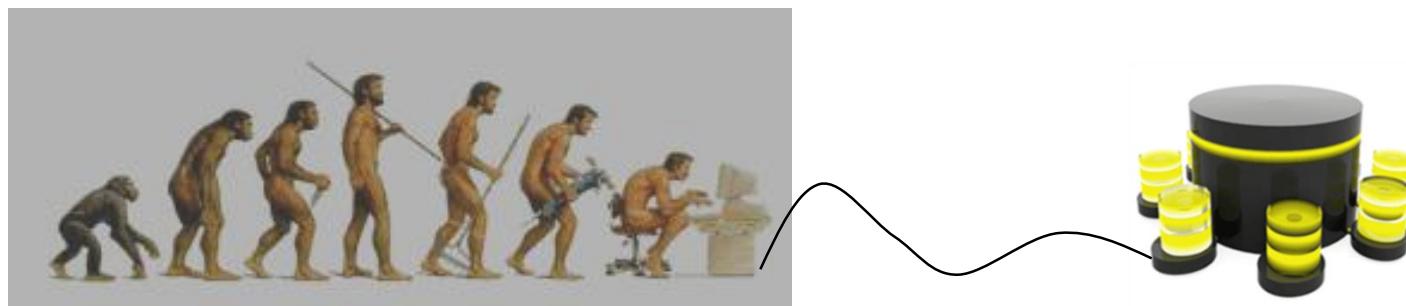
- 다양한 데이터 항목들이 영속적이고 접근자가 여럿인 경우 사용
 - 데이터 생산자와 소비자를 분리
- 성능, 보안, 신뢰성, 기존 데이터와의 호환성에 초점



4.5.1 Shared-Data Style

❖ 다른 스타일과의 관계

- 클라이언트/서버 스타일과 공통되는 부분
 - 다중 클라이언트/서버 형태
- 발행구독 스타일의 블랙보드 스타일과 유사
 - 블랙보드는 데이터의 영속성이 없음
- 데이터 모델 스타일과의 관계
 - 데이터 모델 : 데이터 엔티티와 관계
 - 공유 데이터 스타일 : 데이터 저장소와 접근자
- 배치 스타일과의 관계
 - 배치 스타일은 저장소와 다른 컴포넌트의 하드웨어 노드 할당 상태를 보여줌

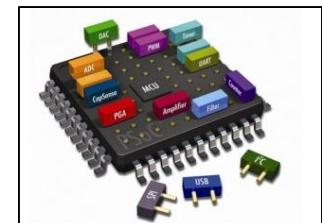
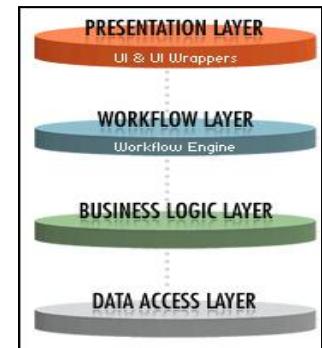
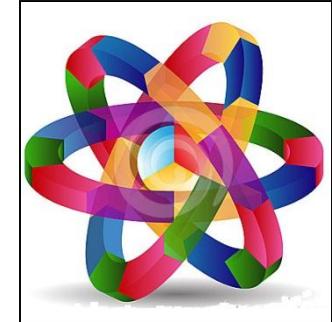


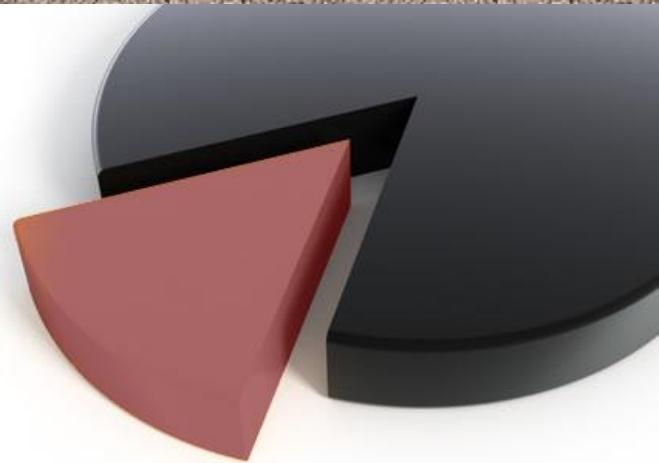


4.6 Crosscutting Issues for C&C Styles

❖ C&C 스타일의 횡단 이슈(Crosscutting Issue)

- 동시성(Concurrency)
 - 시스템 상의 컴포넌트들이 동시에 쓰레드나 프로세스로 실행
- 계층의 사용(Use of Tier)
 - 계층으로 구성 요소 그룹을 통합
 - 접하지 않은 계층 사이의 컴포넌트 간 통신 경로를 제한
- 동적 구조변경(Dynamic reconfiguration)
 - 실행 시 컴포넌트의 생성 및 제거가 가능





Chapter. 5

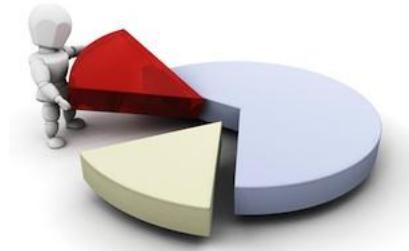
**ALLOCATION VIEWS AND A TOUR OF
SOME ALLOCATION STYLES**



5.1 Overview

❖ 소프트웨어 아키텍처 문서화 시

- 아키텍처와 상호작용 하는 내용도 함께 문서화 해야 한다.
 - 하드웨어, 팀 구조, 파일 시스템 등
 - S/W 아키텍처 ↔ 하드웨어 : 시스템 성능 분석
 - S/W 아키텍처 ↔ 팀 구조 : 효과적인 프로젝트 관리
 - S/W 환경 ↔ 파일 구조 : 시스템 개발 진행



❖ 할당 뷰 (Allocation View)

| | | |
|------------------------|---|---|
| 개요 (Overview) |  | <ul style="list-style-type: none">▪ 할당 스타일들은 소프트웨어 아키텍처와 환경 사이를 매핑 |
| 요소 (Elements) |  | <ul style="list-style-type: none">▪ 소프트웨어 요소와 환경적인 요소▪ 소프트웨어 요소는 환경에서 요구하는 특성을 가진다.▪ 환경 요소는 소프트웨어에 제공해야 하는 특성을 가진다. |
| 관계 (Relations) |  | <ul style="list-style-type: none">▪ Allocate-to : 소프트웨어 요소는 환경 요소와 매핑(할당)된다.▪ 특성은 특정 스타일에 의존한다. |
| 제약 사항 (Constraints) |  | <ul style="list-style-type: none">▪ 스타일에 따라 다양하다. |

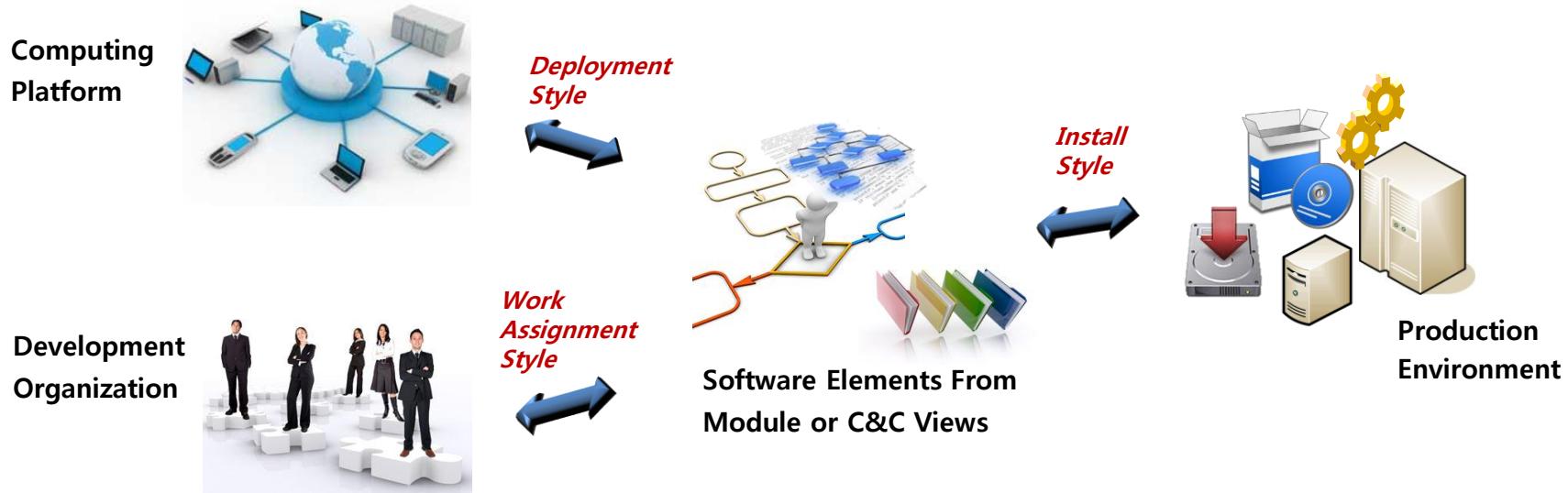


5.1 Overview

❖ 할당 뷰 (Allocation View)

- 모듈 스타일이나 C&C 스타일에 나오는 요소와 환경적 요소를 대응

| | | |
|-------------------------------------|--|---|
| 배치 스타일 (Deployment Style) | | <ul style="list-style-type: none">▪ 소프트웨어가 실행될 하드웨어에 컴포넌트와 커넥터가 어떻게 대응되는지를 설명 |
| 구현 스타일 (Implementation Style) | | <ul style="list-style-type: none">▪ 모듈을 담은 파일 시스템에 모듈이 어떻게 대응되는지를 설명 |
| 작업할당 스타일 (Work Assignment Style) | | <ul style="list-style-type: none">▪ 모듈을 구현하는 사람, 그룹, 팀에 모듈이 어떻게 대응하는지를 설명 |





5.2 Deployment Style

❖ 요소, 관계, 속성

| | |
|--------------------------------|--|
| 개요 (Overview) |  <ul style="list-style-type: none">배치 스타일(Deploy Style)은 소프트웨어 아키텍처의 Component와 Connector를 컴퓨팅 플랫폼의 하드웨어에 매핑 한다. |
| 요소 (Elements) |  <ul style="list-style-type: none">소프트웨어 요소 : C&C 뷰의 요소<ul style="list-style-type: none">하드웨어로부터 요구되는 중요한 특성을 포함한 문서화를 위한 유용한 특성들<ul style="list-style-type: none">프로세싱, 메모리, 용량 요구사항, 결함 방지 등환경 요소 : 컴퓨팅 프로세서의 하드웨어 요소<ul style="list-style-type: none">할당 결정에 영향을 주는 하드웨어 측면의 환경적 요소들의 중요한 특성들<ul style="list-style-type: none">프로세서, 메모리, 디스크, 네트워크 |
| 관계 (Relations) |  <ul style="list-style-type: none">Allocated-to (할당) : 소프트웨어 요소가 어떤 물리적 장치에 탑재되는지 보여준다.<ul style="list-style-type: none">특성은 할당이 실행 시간에 변경 가능한지 여부를 포함Migrates-to (이동), Copy-migrates-to (복사 이동), Execution-migrates -to (실행 이동) : 할당이 동적으로 이루어지는 경우에 사용 |
| 제약 사항 (Constraints) |  <ul style="list-style-type: none">할당의 구성 형태(Allocation Topology)에 대한 제약이 없다.<ul style="list-style-type: none">하드웨어에 의해 제공되는 특성에 의해 소프트웨어에 요구되는 특성은 반드시 만족되어야 한다. |

5.2 Deployment Style

❖ 배치 스타일에서의 관계

- 일반적으로 “할당(Assignment to)”
 - 할당은 소프트웨어 요소가 어떤 물리적 장치에 탑재되었는지 보여준다.
 - 시스템 동작 시 할당 내용이 변경될 수 있으므로 할당은 동적일 수 있다.
 - 추가적인 관계
 - **Migrates-to (이동)**
 - ▶ 임의의 프로세스에서 동작하는 소프트웨어 요소를 다른 프로세스로 옮김
 - ▶ 소프트웨어 요소는 동시에 양쪽 프로세스에 존재하지 않는 경우
 - **Copy-migrates-to (복사 이동)**
 - ▶ 이동과 유사하지만, 원래 프로세스에 소프트웨어 요소가 남아 있으면서
 - ▶ 자신을 복사한 사본을 새로운 프로세스로 보냄
 - **Execution-migrates-to (실행 이동)**
 - ▶ 복사 이동과 유사, 프로세스간 이동은 하지만, 코드가 옮겨가지 않는 관계
 - ▶ 한 프로세스가 여러 프로세서에 존재 가능하지만 특정 시점에 실행되는 프로세스는 단 하나뿐인 경우



5.2 Deployment Style

❖ 배치 스타일 요소의 특성

- 소프트웨어를 물리적인 요소에 할당할 때 영향을 주는 속성이 중요
 - 물리적인 요소가 S/W 요소의 요구사항을 만족시킬지는 S/W와 H/W 속성을 모두 봐야 가능
 - 분석 형태에 따라 요소들이 갖추어야 할 속성값이 결정될 수 있음
 - 메모리 용량 분석이 필요 → S/W 요소는 메모리 소비 측면의 속성을 갖춰야 함



| | | |
|---------------------|----------------|------------------------------------|
| 하드웨어 요소와 관련된 속성 | CPU 속성 | ▪ 연산 요소와 관련 된 속성, 클럭 속도, 프로세스 개수 등 |
| | 메모리 속성 | ▪ 메모리 관련 속성, 메모리 용량, 메모리 속도 등 |
| | 디스크나 저장 장치의 용량 | ▪ 저장 용량과 접근 속도, 등 |
| | 대역폭 | ▪ 통신 채널의 데이터 전송 능력 |
| | 결합 허용 | ▪ 장애 처리 제어 방식에 대한 속성 |
| 소프트웨어 요소와 관련된 속성 | 자원 소비 | ▪ 연산에 필요한 명령어 수 등 |
| | 자원 요구사항과 제약사항 | ▪ 실행에 대한 시간 제약 사항 등 |
| | 안전 중요성 | ▪ 소프트웨어의 동작 시간에 대한 속성 등 |
| 할당과 관련된 속성 | 이동 트리거 | ▪ 시스템 중 할당이 변경되는 경우, 선행되어야 하는 요건 |



5.2 Deployment Style

❖ 배치 스타일의 용도

- 성능, 신뢰성, 보안에 대한 분석과 비용 예측에 활용
 - H/W와 S/W의 할당 구조를 변경함으로써 성능 최적화가 가능
 - 신뢰성은 프로세스 요소나 통신 채널에 직접적인 영향을 받음
 - 시스템 배치 비용은 시스템의 하드웨어 요소에 영향
 - 배치 뷰에서는 개별 구성에 따른 하드웨어 요소 및 용도를 표현해야 함
 - 하드웨어 요소에 배치되는 프로세스
 - 할당을 통하여 기업 전반에 걸친 배치 작업 수행이 가능
 - 배치 스타일 ≠ 소프트웨어 아키텍처
 - 하나의 뷰로 소프트웨어 아키텍처를 완전히 기술할 수 없음
 - 배치 스타일에서의 설계 오류
 - 배치 단위에 추가로 다른 소프트웨어 단위를 넣는 경우에 발생



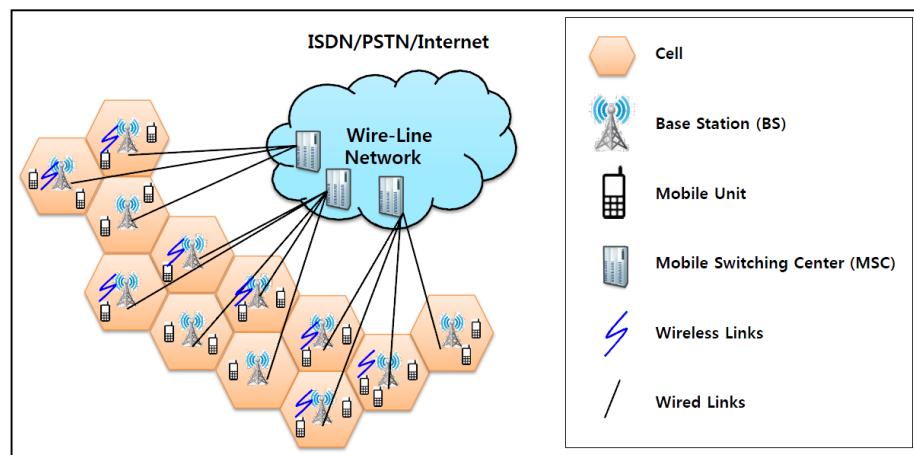


5.2 Deployment Style

❖ 표기법

▪ 비공식적인 표기법

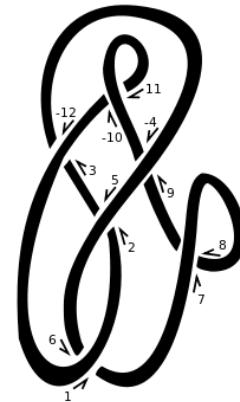
- 도형(S/W, H/W 요소)과 선, 화살표(할당 관계)
 - 양식화 된 기호나 아이콘도 환경적 요소 표현에 사용됨
 - 배치 구조가 단순한 경우 S/W 단위와 관련 H/W 목록을 표로 작성 가능



때때로 비공식적인 표기법이
더 이해하기 쉽고 직관적일 때가 있다.

▪ 정형적 표기법

- AADL (Architecture Analysis and Design Language)
- SysML

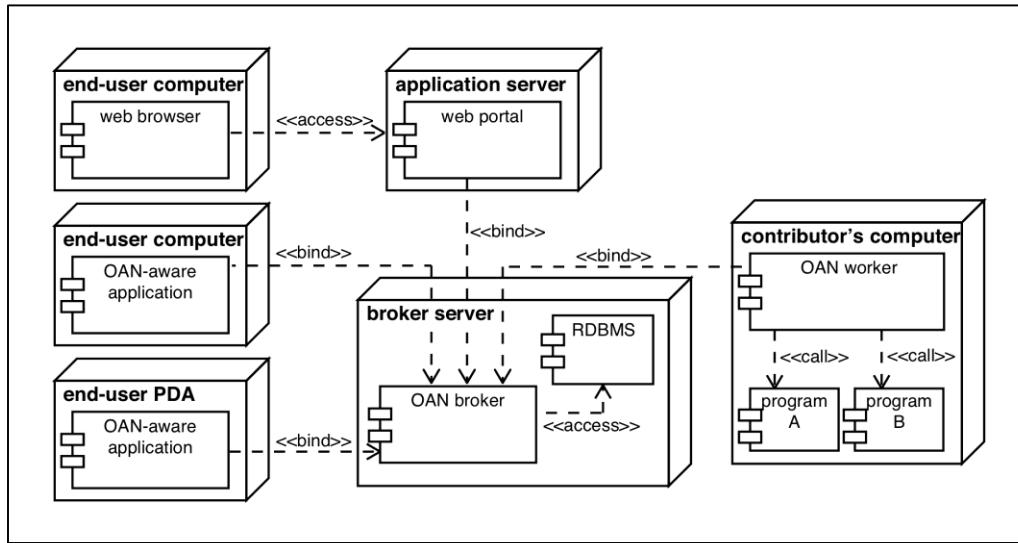




5.2 Deployment Style

❖ 표기법

- UML – Deployment Diagram



배치 다이어그램을 보면 시스템을 구성하는 HW의 연결 관계와 HW에 대한 소프트웨어 컴포넌트의 배치 상태를 알 수 있군 ~



❖ 다른 스타일과의 관계

- C&C 스타일과의 관계

- 배치 스타일은 소프트웨어와 컴퓨팅 플랫폼의 하드웨어의 할당을 보여준다.
- Install Style은 하드웨어 노드에 대한 파일의 할당을 보여준다.



5.3 Install Style

❖ 개요(Overview)

- C&C 스타일의 컴포넌트를 생산 환경의 파일 관리 시스템에 할당
 - 라이브러리, 실행파일, 데이터 파일, 로그 파일, 구성 및 버전 컨트롤 파일 등
- 대규모 소프트웨어 시스템에서 설치되는 파일들은 조직화 될 필요가 있다
 - 시스템 빌드와 패키지 제작 프로세스의 무결성을 유지하고, 통제하기 위해...
 - 배포자와 운영자 필요 시 파일을 찾아 처리하는 것을 돋기 위해...
- 아키텍처 설명(Architecture Description)
 - 설치된 시스템이 파일과 폴더의 구조로 어떻게 구성되었는가를 보여준다.
 - 소프트웨어 요소가 어떻게 파일 구조와 맵핑 되는가를 보여준다.
 - 해당 정보는 개발자, 배포자, 운영자에게는 중요한 자산
- 시스템의 여러 버전을 생산하기 위한 특정 파일의 사용/구성, 패키징 정보
 - 국제화 지원
 - 다른 가격 정책 (무료버전, 사용버전)
 - 서로 다른 고객을 위한 주문 제작 수용
 - 구 버전의 메시지 요청을 보내는 분산 시스템 내의 고객 지원





5.3 Install Style

❖ 요소, 관계, 속성

- 소프트웨어나 환경적 요소에서 중요한 속성
 - 소프트웨어를 형상 항목에 할당할 때 영향을 줄 수 있는 속성들



| | | |
|--------------------|---|--|
| Overview |  | <ul style="list-style-type: none">▪ 생산 환경의 파일 시스템과 소프트웨어 아키텍처의 컴포넌트 간의 맵핑을 설명 |
| Elements |  | <ul style="list-style-type: none">▪ 소프트웨어 요소 : C&C 컴포넌트▪ 환경적 요소 : 파일이나 폴더와 같은 구성 항목 |
| Relations |  | <ul style="list-style-type: none">▪ 할당 (Allocated-to) : 모듈이 하나의 형상 항목으로 할당되는 것을 나타낸다.▪ 포괄 (Containment) : 다른 형상 항목을 포함하는 형상 항목을 보여준다. |
| Constraints |  | <ul style="list-style-type: none">▪ 파일과 폴더가 트리 구조를 구성된다.▪ 포함관계를 형성 : is-contained-in |



5.3 Install Style

❖ 설치 스타일의 용도

- 빌드와 배치 절차 생성
- 설치된 시스템을 구성하는 많은 수의 파일과 폴더를 탐색
 - 특정 파일을 위치 지정에 대해서는 주의가 필요 (로그 파일 또는 환경 설정 파일)
- 소프트웨어 제품 라인의 특정 버전을 만들기 위한 구성 파일의 선택
- 동일 시스템의 다양한 설치 버전의 파일의 갱신과 구성
- 목적이나 생산에 문제를 빠진 내용이나 손상된 파일을 정의
- 자동 갱신 특성의 설계와 구현
- 구입 옵션의 분석을 지원하는 데 사용

❖ 표기법

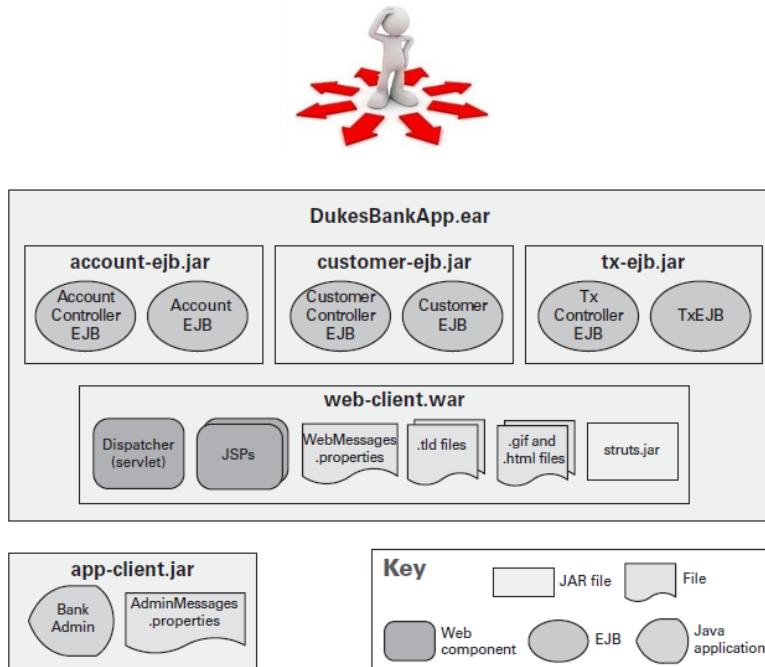
- 컴포넌트, 파일과 폴더 사이의 매핑 관계를 보여준다.
- UML에서 지원되는 스테레오 타입
 - <<artifact>>, <<manifest>>



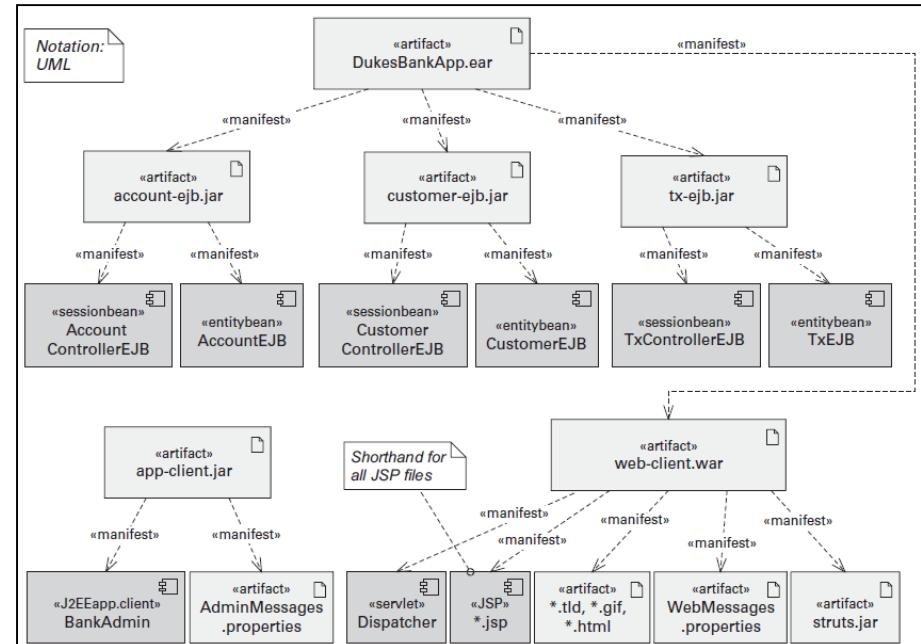


5.3 Install Style

❖ 표기법 (cont')



[비공식적인 표기법]



[UML 표기법]

❖ 다른 스타일과의 관계

- C&C 스타일과 강한 연관 : 소프트웨어 요소의 할당을 설명
- 배치 스타일 : 하드웨어 요소와 설치된 파일의 관계를 보여줌



5.4 Work Assignment Style

❖ 개요(Overview)

- 모듈 스타일에서의 모듈을 구현 책임이 있는 개인이나 그룹에 할당
- 개발팀의 모듈의 구현 및 통합에 대한 책임을 정의
 - Ex) WBS(Work Breakdown Structure)



❖ 요소, 관계, 속성

| | |
|------------------------|--|
| Overview | <ul style="list-style-type: none">▪ 소프트웨어 아키텍처와 개발 조직내의 팀들간의 매핑을 나타낸다. |
| Elements | <ul style="list-style-type: none">▪ 소프트웨어 요소 : 모듈▪ 환경적 요소 : 조직 구성 단위 (사람, 팀, 부서, 등..) |
| Relations | <ul style="list-style-type: none">▪ 할당 (Allocated-to) : 하나의 소프트웨어 요소가 하나의 조직 단위에 할당 |
| Constraints | <ul style="list-style-type: none">▪ 할당에 대한 제한이 없다. 그러나 일반적으로 하나의 모듈은 하나의 조직단위에 할당하는 것으로 제한한다. |



5.4 Work Assignment Style

❖ 작업 할당 스타일의 용도

- 동작하는 시스템을 구성하기 위해 주요 소프트웨어 단위들이 표현된다.
 - 주요 소프트웨어 단위들을 누가 만들어야 하는지를 보여준다.
- 자원 할당과 구축의 책임, 프로젝트 구조의 설명을 돋는다.
 - 작업 할당 구조와 예산/일정의 추정의 기초

❖ 표기법

- 작업 할당 구조 뷰를 위한 특별한 표기법은 없다.

| ECS Element (Modules) | | Organizational Unit |
|--|-------------------------|-------------------------|
| Segment | Subsystem | |
| Science Data Processing Segment (SDPS) | Client | Science team |
| | Interoperability | Prime contractor team 1 |
| | Ingest | Prime contractor team 2 |
| | Data Management | Data team |
| | Data Processing | Data team |
| | Data Server | Data team |
| | Planning | Orbital vehicle team |
| Flight Operations Segment (FOS) | Planning and Scheduling | Orbital vehicle team |
| | Data Management | Database team |
| | User Interface | User interface team |
| ... | ... | ... |

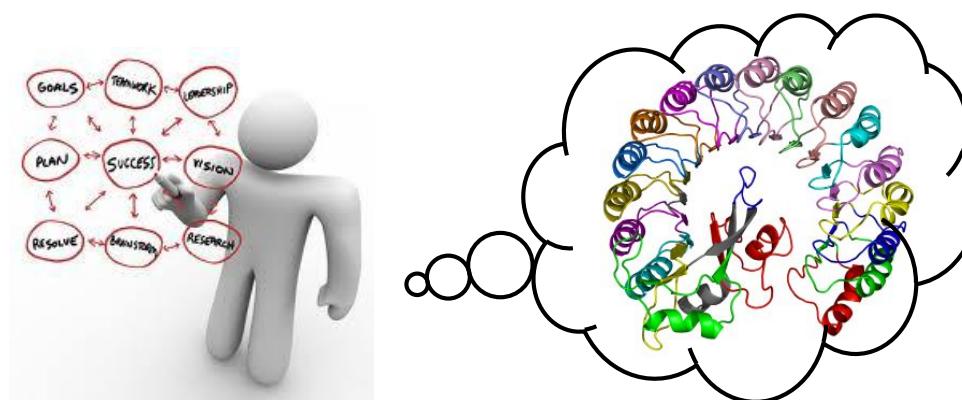
ECS : NASA의 사용 -
작업 할당 뷰를 테이블
형식으로 표현



5.4 Work Assignment Style

❖ 다른 스타일과의 관계

- 작업 할당 스타일은 분해 스타일(Decomposition Style)과 관련
 - 작업의 할당과 매핑의 일반적인 기초가 되기 때문
- 모듈 분해로 확장 가능
 - 개발 도구나, 테스트 툴, 형상 관리 시스템 등과 관련되는 모듈을 추가할 경우
- 다른 뷰와 조합 될 수 있다.
 - 예제: 계층 구조로 내에서의 계층 내 작업 할당 표시
 - 이 경우 도구의 구축에 대한 부분이 빠질 수 있다.
 - 많은 경우, 보조적인 소프트웨어 도구는 시스템의 일부가 아니다.
- 작업 할당 뷰를 생성 시 관리 가능한 크기로 작업을 분해 방법에 대해 주의 필요





5.5 Other Allocation Styles

| | |
|---|---|
| 구현 스타일 (Implementation Style) | <ul style="list-style-type: none">개발 환경에서 파일과 폴더의 트리 구조를 보여준다.<ul style="list-style-type: none">- 개발자가 산출물을 탐색하거나 개발 산출물을 적절한 곳에 위치시키도록 돕는다.설치 스타일(Install Style)이 생산 환경에서의 파일과 폴더의 구조를 보여준다면, 구현 스타일은 개발 환경에서의 파일과 폴더의 구조를 보여준다. |
| 데이터 저장소 스타일 (Data Stores Style) | <ul style="list-style-type: none">SW 데이터 엔티티와 소프트웨어가 위치하는 데이터 서버의 HW 간의 매핑을 보여줌<ul style="list-style-type: none">- 데이터 베이스의 지리적인 분산이나 복제를 보여준다.- 데이터 웨어하우스가 위치하는 머신과 데이터 저장소를 보여줄 수 있다.배치 스타일과 비슷하나, 데이터 엔티티와 HW간의 매핑을 보여준다는 점이 다르다. |
| 요구사항 할당 스타일 (Requirement-Allocation Style) | <ul style="list-style-type: none">시스템 요구사항과 소프트웨어 아키텍처 요소간의 매핑을 보여준다. |
| 배치스타일의 특화 (Specializing the deployment Style) | <ul style="list-style-type: none">배치 스타일은 내제하는 토플러지에 대한 제한이 없지만 특별하게 유용한 배치 스타일의 유형을 발견할 수 있다.<ul style="list-style-type: none">- MS의 Tiered Distribution 패턴, IBM의 WebSphere 의 11개 토플러지 |
| 작업 할당 스타일의 특화 (Specializing the Work Assignment Style) | <ul style="list-style-type: none">플랫폼 스타일 (Platform Style) : Product Line의 개발 시 사용역량 중심 스타일(Competence-center Style) : 해당 분야의 역량이 있는 팀에 관련 업무 할당오픈 소스 스타일(Open-Source Style) : 많은 독립적인 공헌자를 활용프로세스 단계 스타일 (Process-Step Style) : 개발 프로세스를 여러 사이트에 걸쳐 적용배포 기반 스타일 (Release-based Style) : 제품의 개발과 릴리즈가 사이트를 돌며 진행 |



Part II

BEYOND STRUCTURE: COMPLETING THE DOCUMENTATION



Chapter. 8

DOCUMENTING BEHAVIOR



8. Documenting Behavior

❖ 아키텍처의 행위적 측면 문서화

- 아키텍처 개발과 시스템 유지 보수 모두에 혜택을 제공
 - 시스템의 이해
 - 이해당사자의 품질 요구사항을 만족시키기 위한 아키텍처에 대한 근거 제공
- 구조적 뷰의 추가적인 내용을 기술
 - 아키텍처적 요소들이 해당 구조를 통해 어떻게 상호작용하는가를 설명





8.1 Beyond Structure

❖ 뷰의 행위적 측면을 문서화

- 구조적인 정보에 시간 축을 더해야 함
- 구조적 관계를 바탕으로 시스템 내에서 발생하는 상호작용을 반영
 - 특정 시간 상에서 보면 상호작용 중 일부만 실행
- 시스템 행위
 - 요소간 상호작용이 특정 시간이나 시스템 상태에 끼치는 영향을 설명
 - 일부 시스템 행위는 시스템 구조 설명으로 분석이 가능 (Definition-use analysis)
 - 데드락, 시간 내 작업 완료 등의 판단은 요소 행위와 상호작용의 제약 사항이 있어야 판단 가능
- 행위 문서화는 다음의 정보를 필요로 한다.
 - 요소들 사이의 상호작용 순서
 - 동시성 발생 가능성
 - 상호작용간의 시간 종속성
 - 시스템이나 시스템 일부의 가능한 상태
 - 여러 가지 시스템 자원의 사용 패턴



8.2 How to Document Behavior

| | |
|--|--|
| <p>1단계 해답이 필요한 질문의 종류가 무엇인지 결정하기 Decide What Kinds of Questions You Need to Answer</p> | <ul style="list-style-type: none">설계되는 시스템의 종류의존하고, 개발의 단계, 그리고 설계 노력에 초점을 맞춘 모델화하기 위해 행위의 종류를 결정개발 초기에는 입력 데이터가 출력으로 어떻게 변환되는지에 대한 세부사항보다는 요소들과 요소들이 어떻게 상호작용하는지에 중점요소 내에서 변환 행위가 시스템의 전체 행위에 영향을 주기 때문에 요소내의 변환 행위에 대한 제한사항에 대하여 파악하는 것은 유용최소한 자극 행위에 대한 모델과 한 요소에서 다른 요소로의 정보의 이전에 대하여 모델을 만들어야 한다.문서는 제약사항과 상호작용에 대한 명시적인 정보를 포함해야 한다. |
| <p>2단계 어떤 종류의 정보가 이용 가능한지, 제한될 수 있는지를 결정하기 Determine What Types of Information Are Available or Can be Constrained</p> | <ul style="list-style-type: none">행위 다이어그램은 정보의 전송 측면과 한 요소에서 다른 요소로 자극 행위에 대한 설명을 제공해야 한다.포함되는 특성<ul style="list-style-type: none">✓ 일반적인 목적/행위 방법/적용 범위제약 사항<ul style="list-style-type: none">✓ 순서에 따른 제약사항/시간에 기반한 자극 |
| <p>3단계 표기법 선택하기 Choose a Notation</p>  | <ul style="list-style-type: none">시스템 행위에 대한 문서화를 지원하는 모든 언어는 반드시 상호작용의 시퀀스를 묘사할 수 있어야 한다.시퀀스는 시간에 따른 순서이며 이것은 시간 기반의 종속성을 보여줄 수 있다.상호작용과 실행되는 활동들의 시퀀스는 임의의 자극이 도착한 다음에 일어나는 순서로 표시된다.행위 문서화에서 발견되는 표현들<ul style="list-style-type: none">✓ 자극과 활동, 상호작용의 순서, 행위와 연결되는 관계의 구조적 요소2가지 종류의 행위 문서화 방법<ul style="list-style-type: none">✓ 시나리오 동안의 시스템의 구조적 요소를 통해 일어나는 것을 문서화(traces)✓ 구조적 요소나 요소 셋의 완전한 행위를 상태 기반으로 문서화 (comprehensive model) |



8.3 Notation for Documenting Behavior

❖ Trace 표기법

| Use case diagram | Sequence diagram | Communication diagram | Activity diagram |
|------------------|------------------|-----------------------|------------------|
| | | | |
| State chart | Timing diagram | SDL | BPEL |
| | | | |

SDL : Specification and Description Language

BPEL : Business Process Execution Language



8.3 Notation for Documenting Behavior

❖ Comprehensive Model 표기법

- 구조적 요소들의 완전한 행위를 표현한다.
- 초기 상태에서 최종 상태까지의 모든 가능한 경로를 파악할 수 있다.
- 상태 머신 언어
 - 제약사항과 함께 시스템 요소의 구조적인 기술이 가능
 - 외부적 또는 환경적 자극의 일정 시간 내의 반응을 표현 가능

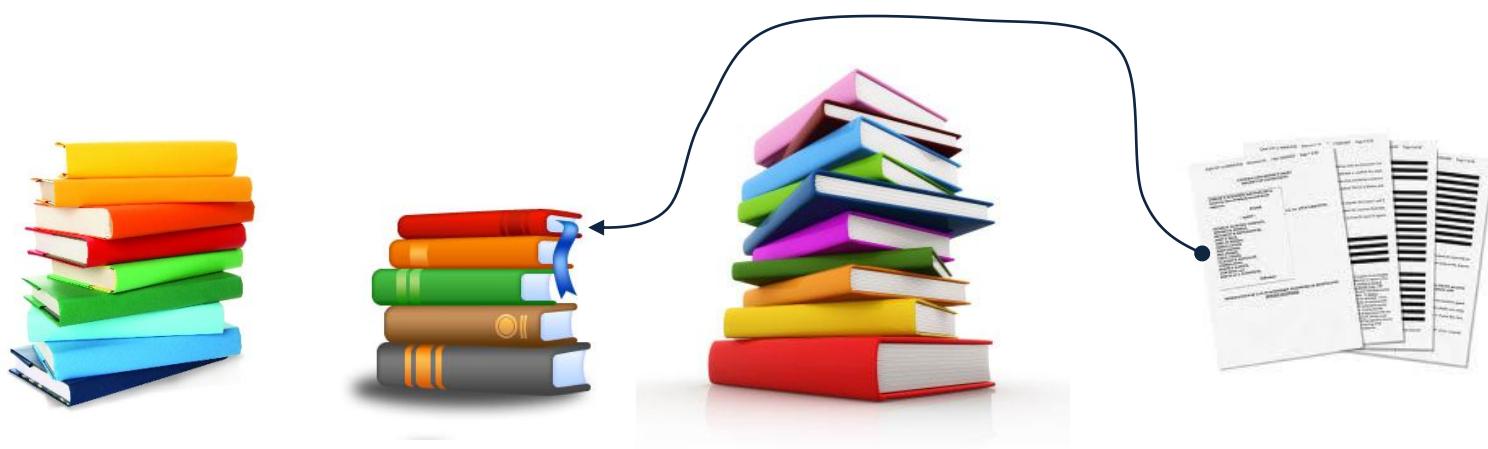
| State machine diagram | AADL |
|--|--|
| <pre>graph LR; Start(()) --> Proposed[Proposed]; Proposed --> Scheduled[Scheduled]; Scheduled --> OpenForEnrollment[Open For Enrollment]; OpenForEnrollment --> Full[Full]; OpenForEnrollment --> ClosedToEnrollment[Closed to Enrollment]; Full --> ClosedToEnrollment; ClosedToEnrollment --> BeingTaught[Being Taught]; BeingTaught --> FinalExams[Final Exams]; ClosedToEnrollment -- "student dropped [seminar size > 0]" --> FinalExams; ClosedToEnrollment -- "term started" --> BeingTaught; BeingTaught -- "closed" --> FinalExams; BeingTaught -- "student dropped [seminar size = 0]" --> FinalExams;</pre> | <p>system type1 end type1;</p> <p>system type2 extends type1 end type2;</p> <p>system implementation type1.impl1 end type1.impl1;</p> <p>system implementation type1.impl2 extends type1.impl1 end type1.impl2;</p> <p>system implementation type2.impl1 end type2.impl1;</p> <p>system implementation type2.impl2 extends type1.impl2 end type2.impl2;</p> <pre>graph TD; type1 --> type1.impl1; type1 --> type1.impl2; type2 --> type2.impl1; type2 --> type2.impl2; type1.impl1 --> type1.impl2; type2.impl1 --> type2.impl2;</pre> |

AADL : Architecture Analysis and Description Language

8.4 Where to Document Behavior

❖ 행위 문서화의 위치

- 행위를 아키텍처 문서 패키지의 어느 부분에 기록할지는 내용에 따라 다르다.
 - 요소가 특정 방법으로 자극을 받을 경우 어떻게 작동하는지를 보여주고 싶을 때
 - 전체 시스템의 포함 요소의 집합이 서로 어떻게 상호작용하는지를 보여주고 싶을 때
- 뷰 개괄 문서
 - 아키텍처가 어떻게 요구사항을 만족시키는지에 대한 논리적 근거부분
→ 설계 정당성의 일부로 삽입 가능

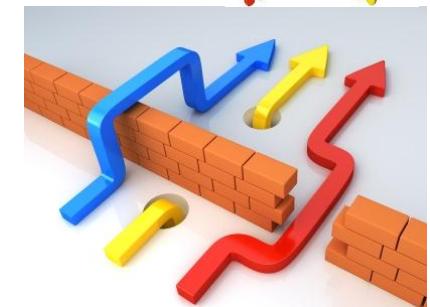




8.5 Why to Document Behavior

❖ 행위 문서화의 필요성

- 시스템의 행위 문서는 이해당사자들 사이의 의사소통을 위해 사용된다.
 - 개발과정과 유지보수 과정에서 사용
- 개발 활동의 추진
 - 이해당사자와의 중요한 의사소통 수단
 - 시스템 요소의 내부 행위와 시스템의 전체 구조의 이해를 촉진
 - 시스템 모의 시험에 활용
 - 시스템 테스트 케이스 개발에 활용 가능
- 시스템 분석
 - 행위를 문서화하면 시스템의 완결성과 정확성, 품질 속성에 대한 논리적 판단에 도움
 - 시스템 요구사항의 충족 여부 확인
 - 시스템 기능의 일관성 확인
 - 성능, 안정성, 변경용이성 예측





Part III

BUILDING THE ARCHITECTURE DOCUMENTATION



Chapter. 11

REVIEWING AN ARCHITECTURE DOCUMENT



In this Chapter ...

❖ 이 장의 목적은

- 아키텍처 평가가 아닌 아키텍처의 문서화를 평가하는 것이다.



❖ 목적의 적합성을 위한 문서화 리뷰

- 문서가 올바른 정보를 적절한 방법으로 표현하고 있는지 확인
 - 이번 장에서는 문서 리뷰를 위한 절차를 설명한다.
- 체크리스트는 문서화 산출물이 적절한지를 평가한다.





11.1 Steps of the Procedure

❖ 문서 리뷰 절차

1

검토 목적 수립하기 Establish the purpose of the review

2

검토 대상 설정하기 Establish the subject of the review

3

적절한 질문 세트 구성하기 Build or adapt the appropriate question set(s)



4

리뷰의 세부 사항 계획하기 Plan the details of the review

5

리뷰 수행 Perform the review

6

결과를 분석하고 요약하기 Analyze and summarize the results

11.1 Steps of the Procedure

❖ 1단계 - 검토 목적 수립하기

- 이해당사자에 의해 설정된 특정 목적에 부합하는 검토 목적 수립
- 리뷰 참가자와 직접적인 리뷰에 중점
- 아키텍처 문서는 하나 이상의 목적에 부합
 - 다원적인 리뷰가 될 수 있다.
 - 한가지 목적을 가진 여러 개의 작은 리뷰들은 대안이 될 수 있다.

Why?



- 검토 목적 수립 시,
 - 문서의 리뷰의 이해당사자를 정의해야 한다.
 - 언제 리뷰를 할 것인지 결정해야 한다.
 - 어떤 수명주기 프로세스를 사용하든 문제가 되지 않는다.
 - 리뷰 목적에 따라 프로젝트 단계 또는 마일스톤과 연관된다.

Who?

When?



11.1 Steps of the Procedure

❖ 개발주기와 아키텍처 리뷰



| 개발 단계 | 활동 | 아키텍처 설계 리뷰 |
|-------------|--|--|
| Concept | <ul style="list-style-type: none">■ 이해당사자의 요구사항 정의■ 개념, 목적, 다양한 해결책 탐색■ 대안적인 아키텍처 분석■ 아키텍처적인 개념 준비■ 계약 협상의 일부분으로 고객과 개발자 사이의 의견 교환 | <ul style="list-style-type: none">■ 올바른 이해당사자와 관심 사항의 포착■ 제안에 대한 지원 |
| Development | <ul style="list-style-type: none">■ 시스템 요구사항 정제하기■ 해결책에 대한 설명서 만들기■ 시스템 또는 시스템들을 구축■ 시스템에 대한 V&V (Verifying and Validating) | <ul style="list-style-type: none">■ 규범 사양에 적합성에 대한 지원■ 평가 지원■ 개발 지원■ 생성 및 분석 도구에 입력 지원■ 아키텍처에 대한 구현 적합성 심사 지원■ 프로젝트 계획에 대한 지원 |
| Utilization | <ul style="list-style-type: none">■ 사용자 요구를 충족시키기 위한 시스템의 운영 | <ul style="list-style-type: none">■ 운영 오류에 대한 추적성 지원 |
| Support | <ul style="list-style-type: none">■ 지속적인 시스템 기능 제공 | <ul style="list-style-type: none">■ 아키텍처와 관련된 시스템 진화와 진화를 위한 비즈니스 계획과 관련된 지원 |

11.1 Steps of the Procedure

❖ 2단계 - 검토 대상 설정하기

- 요구되는 산출물의 집합을 정의하고, 리뷰를 위하여 해당 산출물을 모은다.
 - 산출물 탑입, 산출물의 버전과 소스
 - 리뷰를 수행하기 위해 필요한 산출물의 완성도 정도
 - 아키텍처 문서 (AD)는 반드시 필요 !!!
- 모든 검토자가 동일한 산출물의 같은 버전을 검토하도록 해야 한다.

❖ 3단계 - 적절한 질문 세트 구성하기

- 아키텍처 리뷰를 위한 질문 세트 정의
 - 기존 질문 세트가 있는 경우 재사용이 가능
 - 리뷰 목적에 맞도록 조정이 필요할 수 있음
 - 새로운 질문 세트를 만드는 경우
 - 재사용이 가능하도록 만들어야 한다.
 - 리뷰 목적에 맞는 문맥적인 정보와 이해당사자의 요구를 제공
 - 결과를 획득하고, 해석할 수 있는 가이드라인 제공
- 일반적인 질문은 프로젝트의 기술에 따라 특화된 질문으로 교체 가능
- 질문들은 적절한 형식을 갖추도록 할 수 있다.



11.1 Steps of the Procedure

❖ 4단계 - 리뷰의 세부 사항 계획하기

- 리뷰 날짜, 리뷰 시간표(time frame), 리뷰 형식 등
 - 질문의 우선 순위에 따라 시간을 더 할당하거나 제한이 가능하다.
- 실제 리뷰 참가자 지정하고, 그들의 참석 여부를 확보
 - 검토자에게 초기 질문을 할당
 - 이해당사자는 질문에 대한 응답을 할 수 있음
 - 리뷰가 시작되면, 우선 순위와 이해당사자 할당이 변경될 수 있다.
 - 문서에 대한 이해도를 높이기 위해
 - 리뷰어가 적용 가능한 부분을 찾기 위해
- 리뷰를 위한 실행 계획을 조정
 - 회의 장소와 시간
 - 모든 참석자의 참석 여부
 - 미리 읽어야 할 자료 제공





11.1 Steps of the Procedure

❖ 5단계 - 리뷰 수행

- 리뷰에 포함된 이해당사자에게 질문하기
 - 이해당사자가 검토자 역할을 하고, 질문을 할 수 있다.
 - 분리된 팀들이 이해 당사자에게 질문할 수 있다.
- 리뷰 방식은
 - 개인 리뷰, 면대면, 온라인 분산/원격 회의
- 리뷰어가 좀 더 신중히 검토해야 할 질문들이나 산출물
 - 리뷰 초기 부분에 결정해야 한다.



❖ 6단계 - 결과를 분석하고 요약하기

- 질문에 대한 답변을 종합
- 아키텍처 문서의 전반적인 영향에 대한 질적인 결정
 - 결과는 Pass/Fail 과 같이 단순할 수도 있고
 - 아키텍처 문서의 특정 부분의 문제에 집중한 미묘한 답변이 될 수도 있다.





11.2 Sample Question Sets for Reviewing for AD

❖ Question Set Template

| | | |
|----------------|--|---|
| 1 | 질문 세트 이름 (Question Set Name) | <ul style="list-style-type: none">▪ 질문 세트를 재사용 할 때, 참조할 수 있는 질문 세트의 이름 |
| 2 | 목적 (Purpose) | <ul style="list-style-type: none">▪ 질문 세트를 나오도록 한 리뷰의 목적 |
| 3 | 이해당사자와 관심사항 (Stakeholders and Concerns) | <ul style="list-style-type: none">▪ 누가 이해당사자이고, 그들의 관심 사항은 무엇인지를 설명<ul style="list-style-type: none">- 아키텍처 문서 검토의 첫째 수준의 이해당사자와 관심 사항의 설정은 질문 세트의 목적을 효과적으로 정제하고, 질문의 계통적 서식을 알게 한다. |
| 4 Questions | a. 응답자 (Respondents) | <ul style="list-style-type: none">▪ 누구에게 질문을 해야 하는가? |
| | b. 예상 답변 (Expected Answer) | <ul style="list-style-type: none">▪ 찾고자 하는 답변은 무엇인가? |
| | c. 중요도 (Criticality) | <ul style="list-style-type: none">▪ 각 질문들이 얼마나 중요한가? |
| 5 | 조언 사항 (Advice) | <ul style="list-style-type: none">▪ 언제, 어떻게 리뷰가 진행되어야 하는지에 대한 추가적인 유용한 정보를 제공 |



11.2 Examples

❖ Example Question Set for Reviewing for Conformance to ISO/IEC 42010

1. Question Set Name: Reviewing for conformance to ISO/IEC 42010

2. Purpose

This question set is used to assess the conformance of the AD to the requirements of the international standard ISO/IEC 42010. Conformance to the standard may be a prerequisite to acceptance of the AD as a deliverable or to other reviews.

3. Stakeholders and Concerns

Architects, acquirers, and architecture analysts all have the following concern: Does my AD meet all of the conformance points of the standard? Can conformance be verified?

4a. Questions

Respondents: Architects

1. Does the AD contain the appropriate administrative and overview data (date of issue, version status, issuing organization, change history, summary, scope, context, glossary, and references)?
2. Does the AD contain architecture documentation required by the using organization?
3. Who are the specific stakeholders for this AD? Is there evidence the architect has given consideration to these stakeholder classes: users of the system, system acquirers, system developers, and system maintainers?
4. Are the stakeholders' concerns captured? Does the AD show evidence of having considered the purposes of the system; the suitability of the architecture to achieve those purposes; the feasibility of constructing and deploying the system; the potential risks of system to its stakeholders throughout its life cycle; and the maintainability and evolvability of the system?
5. Is every stakeholder and every concern covered by at least one viewpoint?
6. Is each viewpoint identified? Is there a definition for each viewpoint used in the AD? Does each viewpoint definition include: viewpoint name; identification of the stakeholders addressed by that viewpoint; the architectural concerns framed by that viewpoint; and the model kinds used by the viewpoint? For each model kind, are the conventions, including any notations, languages, modeling techniques, and analytical methods, defined?
7. If the viewpoint comes from an external source, is it fully defined and identified in that source? Is there an association between that viewpoint and the stakeholders' concerns? Are models/modeling techniques identified? Does the viewpoint contain analysis techniques, rules, or constraints?
8. Is there a view for each viewpoint? Does the view correctly use/implement the models required by its viewpoint? Does the view cover the system under review? Is the view-viewpoint relationship one-to-one?

9. Does each view contain an identifier, introductory information, configuration information as defined by the using organization, and one or more models?

10. Are any known inconsistencies between views documented?
11. Are there correspondence rules? For each such rule, is there at least one correspondence satisfying each rule?
12. Does the AD cite an existing architecture framework? Is each viewpoint in the framework used in the AD? Does the AD capture all of the framework's correspondence rules?
13. Does the AD contain the rationale for its architectural decisions, such as
 - Selection of viewpoints and models/modeling techniques?
 - Correspondence rules?
 - Key decisions captured within each view?

Respondents: Acquirers and architecture analysts

14. Is the set of stakeholders and concerns complete?
15. Is the set of viewpoints both complete and minimal?
16. Is the set of correspondence rules (if used) appropriate?
17. Are the views complete? Do they communicate the key decisions?
18. Is the set of correspondences complete?
19. Does the rationale capture sufficient information to assist reviewers and architecture analysts in understanding the architecture and its decisions?
20. Do the set of viewpoints and/or the selected architecture framework match contractual requirements and/or institutional practices?

4b. Expected Answers

Positive answers are expected, as well as the ability of the participants to point out specific places in the AD to justify their positive answers.

4c. Criticality

For the purpose of ascertaining conformance, all requirements in ISO/IEC 42010 are of equal importance, and all are mandatory. (There are no tailoring options in the standard.)

5. Advice

"Complete" here is expected to be a value judgment in the review, rather than any formally determined property. The stakeholders need to understand the context (including resource constraints) as part of evaluating "completeness." Generally, "complete" should be interpreted as "good enough to meet our expectations for this system within the context in which we are developing it." These rules should not be required as having the architecture description account for every (software equivalent of a) nail in the structure. Each item chosen above directly maps to conformance points in ISO/IEC 42010:2007. However, the terms in this section are taken directly from ISO/IEC FCD 42010:2010.



Q&A ...

