최종연구보고서	KISA-WP-2007-0045

소프트웨어 아키텍처 기반 설계 모델 및 명세기법 개발

Development of Software Architecture-based
Design Model and Architecture Specification Method

수탁기관: 숭실대학교

2007. 12

한국정보보호진흥원
Korea Information Security Agency

제 출 문

한국정보보호진흥원 원장 귀하

본 보고서를 “소프트웨어 아키텍처 기반 설계 모델 및 명세기법 개발”에 관한 최종연구개발 결과보고서로 제출합니다.

2007 년 12 월 11 일

수탁 기관 : 숭실대학교
연구책임자 : 교 수 김 수 동 (숭실대학교 컴퓨터학과)
참여연구원 : 연 구 원 최 시 원 (숭실대학교 컴퓨터학과)
연 구 원 허 진 선 (숭실대학교 컴퓨터학과)
연 구 원 전 원 영 (숭실대학교 컴퓨터학과)
연 구 원 이 재 유 (숭실대학교 컴퓨터학과)

요 약 문

1. 제목

소프트웨어 아키텍처 기반 설계 모델 및 명세기법 개발

2. 연구개발의 목적 및 중요성

1) 연구의 목적

본 연구의 목적은 다음의 세 가지로 나눌 수 있다. 첫째, 공통평가기준 3.1의 개발(ADV) 요구사항을 소프트웨어 아키텍처 공학 및 설계 품질 관점에서 분석하여, 보다 실효성 있는 평가 기준을 마련하기 위한 기술적 근간을 확보한다. 둘째, 소프트웨어 아키텍처 모델을 보안 설계 및 아키텍처 특성(ADV_TDS, ADV_ARC)에 반영한 특화 모델을 개발하고, 대표적 시스템 유형별 적용할 상세 모델을 제시한다. 셋째, 평가에 사용될 제출물의 명세 기법을 개발하여, 산출물 작성 지침으로 활용하게 한다.

2) 연구의 중요성

공통평가기준에서 개발 관련된 보안 설계 (TOE Design, ADV_TDS)와 보안 아키텍처 (Security Architecture, ADV_ARC)는 소프트웨어 공학 관점에서의 평가 프로세스, 평가 기법 및 제출물 작성이 요구되는 분야이다. 공통평가기준에서는 이러한 소프트웨어공학 관점의 세부적인 지침이 포함되어 있지 않으므로, 본 과제에서 이러한 세부 지침을 마련하여 평가에 운영하여야 할 필요가 있다.

3. 연구개발의 내용 및 범위

연구개발 내용	연구개발 범위
소프트웨어 아키텍처 개요 및 특성 분석	- 소프트웨어 아키텍처 소개 - 소프트웨어 아키텍처 설계 프로세스 - 다양한 소프트웨어 아키텍처 모델 분석
공통평가기준 3.1 개발(ADV) 요구사항을 소프트웨어 공학 관점에서 분석	- 보안 구조(ADV_ARC) 요구사항의 3가지 주요 특성인 자체보호(Self-Protection), 우회불가성(Non-bypassability), 영역분리(Domain Isolation)를 소프트웨어 아키텍처 개념을 적용해서 분석 - TOE 설계(ADV_TDS) 요구사항의 각 항목에 대하여, 소프트웨어 공학 측면에서의 설계 요구사항을 반영하여 분석
보안 설계 및 아키텍처 특성(ADV_TDS, ADV_ARC)에 특화된 모델 개발	- 소프트웨어 아키텍처 모델과 ADV_TDS와 ADV_ARC 요구사항간의 연관 관계를 정의 - 시스템의 대표적인 유형별(하드웨어, OS, 응용프로그램 등)로 세분화되고 특화된 모델을 정의
개발된 모델을 이용한 제출물 명세기법 제시	- TOE 설계(ADV_TDS), 보안구조(ADV_ARC) 요구사항과 관련하여 평가를 위한 제출물 작성에 사용될 수 있는 지침 개발

4. 연구결과

2 장에서는 소프트웨어 공학 관점에서 바라본 소프트웨어 아키텍처에 대한 정의와 설계 프로세스, 각 절차 별 설계 지침에 대해 소개한다. 3 장에서는 대표적인 소프트웨어 아키텍처 스타일에 대한 개요, 구조 및 장·단점에 대해 소개한다. 4 장에서는 보안구조(ADV_ARC)가 만족해야 하는 속성인 아키텍처 타당성, 자체보호, 영역분리, 우회불가성에 대해 소프트웨어 공학적인 관점에서 분석한다. 5 장에서는 TOE 설계(ADV_TDS)와 관련해서 정형화의 3 단계, 서브시스템과 모듈, 컴포넌트의 6개 계층 관계에 대해 소프트웨어 공학적으로 분석한다. 6장에서는 보안 아키텍처를 설계할 수 있는 프로세스를 제안하면서, 각 절차마다 ADV_ARC의 3가지 속성에 맞는 지침 및 가이드라인을 제시하였다. 마지막으로 7 장에서는 보안 아키텍처를 효과적으로 명세하기 위하여 템플릿 및 가이드라인을 제시한다.

5. 활용에 대한 건의

본 연구에서 마련될 ADV_TDS, ADV_ARC 기반 평가 프로세스, 평가 기법은 보안 평가

시에 보다 정량적이고 정확한 평가를 수행할 수 있도록 활용한다. 또한, 제출물 작성 지침을 소프트웨어 개발회사에서 적용 준수하도록 하여, 개발 품질 향상과 평가 효율성을 높이는데 활용한다.

6. 기대효과

공통평가기준 3.1에 대한 소프트웨어 아키텍처 및 설계 관점의 분석을 통한 보안설계 및 보안 아키텍처 분야의 국제적 평가 기술을 자체적 확보한다. 본 연구에서 도출될 연구 결과를 통하여 평가 뿐 아니라, 제출물 작성 시 일관성과 표준성을 높인다.

SUMMARY

1. Title

Development of Software Architecture-based Design Model and Architecture Specification Method

2. Purpose of the study

Purpose of the study can be classified into three purposes as the following. The first purpose is to analyze the development (ADV) requirement of common criteria 3.1 from the perspective of software architecture engineering and design quality and to establish the technical basis for developing practical assessment criteria. The second purpose is to develop a customized software architecture model by considering TOE design and security architecture (ADV_TDS, ADV_ARC) and to provide a precise model that can be applied to representative types of system. The third purpose is to develop specification methods for the artifacts that are used in assessment and to use them as the artifact specification guideline.

TOE design (ADV_TDS) and security architecture (ADV_ARC) from the common criteria are the areas that require assessment process, assessment method, and artifact specification method in software engineering viewpoint. Since precise instructions from the perspective of software engineering are not included in the common criteria, we need to develop the precise instructions and apply them in the assessment.

3. Contents and scope

Contents	Scope of the Research Project
Summarizing Software Architecture and Analyzing Characteristics of the Software Architecture	- Introduction to Software Architecture - Software Architecture Design Process - Analysis on Various Software Architecture
Analyzing ADV of CC 3.1 with the Software Engineering Perspective	- Analyzing Three Characteristics of ADV_ARC (Self-Protection, Non-bypassability, and Domain Isolation) with Concepts of Software Architecture - Analyzing ADV_TDS with Design Concepts of Software Engineering
Developing Architecture Model which is Customized to ADV_TDS and ADV_ARC	- Defining How to Map the Software Architecture to ADV_TDS and ADV_ARC - Defining Models Customized to ADV_ARC and ADV_TDS
Proposing Methods to Describe Software Architecture Model	- Developing Guidelines and Templates to Describe Artifacts of Software Architecture and Design

4. Results of the study

Section 2 introduces overall architecture design process and design instructions for each phase with the perspective of software engineering, and section 3 summarizes overview, structure, advantages, and disadvantages for popular software architecture styles. Based on the foundation for the software architecture, section 4 and section 5 analyzes ADV_ARC and ADV_TDS with the software engineering perspectives. Section 7 proposes the software architecture process which is specialized to security architecture, especailly ADV_ARC. And section 8 explaines how to describe security archtiecture in terms of guidelines and templates.

5. Expected effects and applications

By analyzing Common Criteria 3.1 with a perspective of software architecture and software design, we can secure technology for analyzing security architecture in

international level. With the results of this research project, security architecture is fairly evaluated and architecture is described in the consistent and standardized way.

The result of this project can be utilized in evaluating quantitavely and correctly security architecture. And, by letting all the companies comply to guidelines for describing software architecture, we can gain high-quality software development and evaluation efficiency.

목 차

그림 목차

표 목차

Contents

제 1 장 서론

제 1 절 연구 배경

본격적인 정보화 사회에서 소프트웨어의 품질 평가, 특히 보안성 평가에 대한 중요성이 국.내외적으로 더욱 커지고 있다. 지난 10여년에 걸쳐, 개발 요구사항의 복잡도가 크게 증가하면서, 소프트웨어는 점차 대형화되고 있고, 이런 소프트웨어의 보안성 평가를 위해서는 개발 과정 및 이후 과정에서의 품질 평가 기술이 필요하다.

복잡한 소프트웨어의 설계 품질에 큰 영향을 미치는 요소는 아키텍처이며, 이런 중요성을 감안하여, 보안 평가를 위한 표준 기준인 공통평가기준 (Common Criteria for Information Technology Security Evaluation) 3.1이 마련되었다. 이 표준을 준수한 보안 평가 절차와 기법을 확보하여야, 국제적 수준의 보안 평가를 시행할 수 있다.

공통평가기준에서 개발 관련된 보안 설계 (TOE Design, ADV_TDS)와 보안 아키텍처 (Security Architecture, ADV_ARC)는 소프트웨어 공학 관점에서의 평가 프로세스, 평가 기법 및 제출물 작성이 요구되는 분야이다. 공통평가기준에서는 이러한 소프트웨어공학 관점의 세부적인 지침이 포함되어 있지 않으므로, 본 연구에서 이러한 세부 지침을 마련하여 평가에 운영하여야 할 필요가 있다.

제 2 절 연구 범위

1. 소프트웨어 아키텍처 개요 및 특성 분석

가. 소프트웨어 아키텍처 소개

소프트웨어 아키텍처는 소프트웨어를 구성하는 컴포넌트와 컴포넌트간의 관계를 추상적인 수준에서 정의한다. 즉, 소프트웨어 전체 구조를 한눈에 볼 수 있게 해주며, 각 컴포넌트와 컴포넌트 사이의 관계에 대한 이론적 근거를 판단할 수 있게 한다. 소프트웨어 아키텍처의 가치는 소프트웨어 대한 이해를 돕고, 시스템 수준의 설계 모델을 재사용할 수 있게 해주며, 품질 요구사항을 반영할 수 있게 해준다는 것이다. 또한, 소프트웨어 상세 설계 및 구현 이전에 초기 설계 단계에서 소프트웨어가 가질 품질 요구사항을 예측할 수 있게 해준다.

나. 소프트웨어 아키텍처 설계 프로세스

비기능적 요구사항 중 품질 속성이 소프트웨어 설계를 결정짓는 주요한 아키텍처 드라이버가 된다. 아키텍처 드라이버를 기반으로 뷰타입 및 스타일을 선택하여 이를 기반으로 아키텍처를 설계하게 된다. 이 과정에서 아키텍처 드라이버 간에 충돌이 발생하게 되는데 대표적인 소프트웨어 아키텍처 설계 프로세스로인 SEI의 ATAM에서는 품질 속성의 충돌로 인한 설계상의 충돌을 해결하기 위해 민감도(Sensitivity) 식별을 수행한다. 민감도 식별 결과를 기반으로 아키텍처 설계상의 충돌을 해결한다.

다. 다양한 소프트웨어 아키텍처 모델 분석 등

소프트웨어를 바라보는 관점에 따라 대표적으로 정적뷰, 동적뷰, 배치뷰로 소프트웨어 아키텍처를 설계할 수 있다. 각 뷰별로 아키텍처가 요구하는 요구사항에 따라 재사용할 수 있는 아키텍처 패턴을 여러 문헌에서 정의하고 있는데 이를 아키텍처 스타일이라 한다. 아키텍처 스타일은 컴포넌트의 특징, 컴포넌트간의 관계의 특징 및 아키텍처를 설계하기 위한 기타 제약 사항을 정의한다. 현재 제안된 아키텍처 스타일의 종류는 다양하며 대표적인 스타일로는 분해 스타일, 파이트와 필터 스타일, 클라이언트 서버 스타일, 발행 구독 스타일 등이 있다.

2. 공통평가기준 3.1 개발(ADV) 요구사항을 소프트웨어 공학 관점에서 분석

가. 보안 구조(ADV_ARC)에 대한 해석

보안 구조(ADV_ARC) 요구사항의 3가지 주요 특성인 자체보호(Self-Protection), 우회불가성(Non-bypassability), 영역분리(Domain Isolation) 특성을 소프트웨어 아키텍처 개념을 적용해서 분석, 해석한다. 즉, A.1.2에서 명시된 각 특성이 소프트웨어 아키텍처 설계 및 평가의 관점에서 어떻게 관련되어 지는지를 분석, 제시한다.

나. TOE 설계(ADV_TDS)에 대한 해석

TOE 설계(ADV_TDS) 요구사항의 각 항목에 대하여, 소프트웨어 공학 측면에서의 설계 요구사항을 반영하여 분석한다. ADV_TDS.1에서부터 ADV_TDS.6의 기본 설계, 아키텍처 설계, 기본 모듈화 설계, 준 정형적 모듈화 설계, 완성된 준 정형적 모듈화 설계, 정형적 고급 설계 표현을 가진 완성된 준 정형적 모듈화 설계에 대한, 소프트웨어 공학 관점에서의 분석을 한다.

3. 보안 설계 및 아키텍처 특성(ADV_TDS, ADV_ARC)에 특화된 모델 개발

소프트웨어 아키텍처 모델과 ADV_TDS와 ADV_ARC 요구사항간의 연관 관계를 정의하고, 이를 기반으로 시스템의 대표적인 유형별(하드웨어, OS, 응용프로그램 등)로 세분화되고 특화된 모델을 정의한다. 즉, 각 유형의 특징을 고려한 아키텍처의 참조 모델을 정의하고, ARC요구사항 별 관찰 및 평가가 되어야 하는 항목들을 도출한다.

4. 개발된 모델을 이용한 제출물 명세기법 제시

TOE 설계(ADV_TDS), 보안구조(ADV_ARC) 요구사항과 관련하여 평가를 위한 제출물 작성에 사용될 수 있는 지침을 개발한다. 이 지침은 아키텍처 설계와 소프트웨어 설계에 관한 범용적인 가이드라인과 양식(Template)을 포함한다.

제 2 장 소프트웨어 아키텍처의 개요 및 설계 프로세스

본 장에서는 소프트웨어 아키텍처의 개요 및 설계 프로세스에 대해 다룬다. 먼저 소프트웨어 아키텍처의 정의 및 핵심 개념에 대해 소개하고 이를 기반으로 아키텍처 설계를 위한 전체적인 프로세스를 기술하고, 프로세스 내 절차 별 지침사항에 대해 상세히 기술한다. 마지막으로 아키텍처 설계 검증을 위한 대표적인 프로세스 중 하나인 ATAM 프로세스에 대해 설명한다.

제 1 절 소프트웨어 아키텍처 개요

1. 소프트웨어 아키텍처란?

프로그램 혹은 컴퓨팅 시스템의 소프트웨어 아키텍처는 시스템의 구조로서 소프트웨어 요소, 속성, 그리고 이들 간의 관계로 이루어진다 [1].

2. 시스템 구조

가. 정적 구조

정적 구조에서는 설계 시 내부 요소 및 배열을 정의한다. 요소로는 모듈, 객체 지향 클래스, 서비스 등이 될 수 있다. 내부 데이터 요소는 클래스, 관계형 데이터 베이스 개체 및 테이블, 데이터 파일 등을 포함하고 있다. 내부 하드웨어 요소는 컴퓨터이나 컴퓨터 부품, 네트워크 요소 등을 포함하고 있다. 이런 요소들의 배열은 문맥에 따라 달라지는데, 문맥은 요소 간의 연결, 관계 등이 될 수 있다.

나. 동적 구조

동적 구조에서는 시스템이 실제로 어떻게 동작하는지를 보여준다. 즉, 실시간 요소와 요소들 간의 상호 연동을 정의하고 있다. 이런 연동은 병렬 혹은 순차적인 관계로 표현될 수 있다.

3. 시스템 속성

가. 외부로 보이는 행위

시스템과 시스템이 구동되는 환경 사이에서의 기능적인 상호 연동을 정의한다. 이런 상호 연동은 동적인 구조를 위해 고려하는 것과 비슷하다. 이는 시스템에서의 정보 입출력 흐름, 외부 입력에서의 시스템의 반응, API 등을 포함하고 있다. 외부 행위는 시스템을 블랙박스로 간주하고 기술된다.

나. 품질 속성

성능, 보안성, 규모처럼 외부에서 볼 수 있는 시스템의 비기능적인 속성을 뜻한다. 다음과 같은 것들을 고려할 수 있다.

- 작업량에 대해 시스템은 얼마나 성능을 낼 수 있는가?
- 특정 주어진 하드웨어에서 최대 처리량은 얼마인가?
- 악의적인 사용에 대해 시스템은 정보를 어떻게 방어하는가?
- 얼마나 자주 문제가 발생하는가?
- 얼마나 관리, 유지 보수, 보완하기 쉬운가?

4. 아키텍처 요소

아키텍처 요소란 시스템을 구성할 때 고려할 수 있는 기본적인 부분이다. 아키텍처 요소는 다음과 같은 주요 속성을 반드시 갖고 있어야 한다.

- 명확히 정의된 역할의 집합
- 명확히 정의된 경계의 집합
- 명확히 정의된 인터페이스의 집합

5. 이해 당사자

소프트웨어 시스템에 영향 받는 사람은 사용에 국한되지 않는다. 사람들은 소프트웨어 시스템을 만들고, 테스트하며, 고치고, 보완하며, 이에 필요한 금액을 지불한다. 즉, 소프트웨어 아키텍처에서 이해 당사자는 아키텍처 실현화에 대해 관심 있는 사람, 그룹, 혹은 개체를 의미한다. 좋은 이해 당사자의 속성은 다음과 같은 것을 포함한다.

- 좋은 의사 결정을 하기 위해 정보가 충분히 제공된다.
- 건설적인 방향으로 처리나 선택이 가능하도록 자발적으로 참여해야 한다.
- 의사 결정을 하기 위해 권한을 받아야 한다.
- 이해 당사자 그룹이 대표적이어서 그들의 관점이 유효하다는 것을 보여야 한다.

6. 아키텍처 명세서

아키텍처 명세서는 이해 당사자가 이해할 수 있고, 아키텍처가 이해 당사자의 관심을 확실하게 보일 수 있는 방법으로 아키텍처를 문서화해 놓은 산출물들의 집합이다. 여기서 산출물의 범위에는 아키텍처 모델, 범위 정의, 제약사항, 원칙이 포함될 수 있다. 아키텍처 명세서는 아키텍처의 핵심과 그에 관한 자세한 내용을 표현해야 한다. 비록 모든 시스템이 아키텍처를 갖고 있지만, 모든 시스템이 좋은 아키텍처 명세서를 포함하고 있는 것은 아니다. 여기서 좋은 아키텍처 명세서란 효율적이고 일관성 있게 아키텍처의 핵심 측면을 적절한 이해 당사자에게 전달할 수 있도록 기술된 것을 의미한다.

7. 뷰포인트와 뷰

가. 아키텍처 뷰

아키텍처 뷰는 중점을 둬서 언급하고자 하는 관점과 연관있는 아키텍처의 측면이나 요소를 기술하는 방법이다. 그러나 하나의 모델에서 전체 아키텍처의 핵심과 그에 대한 자세한 내용을 담기는 어렵다. 그러므로 4+1 뷰 모델과 같은 뷰 포인트가 필요하다. 최근에는 IEEE 표준 1471에서 이런 개념을 정규화하고 용어를 표준화하였다.

나. 뷰포인트

뷰포인트는 뷰의 형태를 구성하기 위하여 패턴, 양식, 사례를 모은 것이다. 뷰포인트는 뷰포인트에서 관심있는 의도를 반영한 이해 당사자를 정의한다. 또한 그 뷰를 구성하기 위한 지침, 원칙, 양식을 정의한다. 즉 뷰포인트는 재사용할 수 있는 아키텍처 지식을 모아서 프레임웍을 제공함으로써 아키텍터 명세서의 특정 양식을 생성하는데 가이드로 사용될 수 있다.

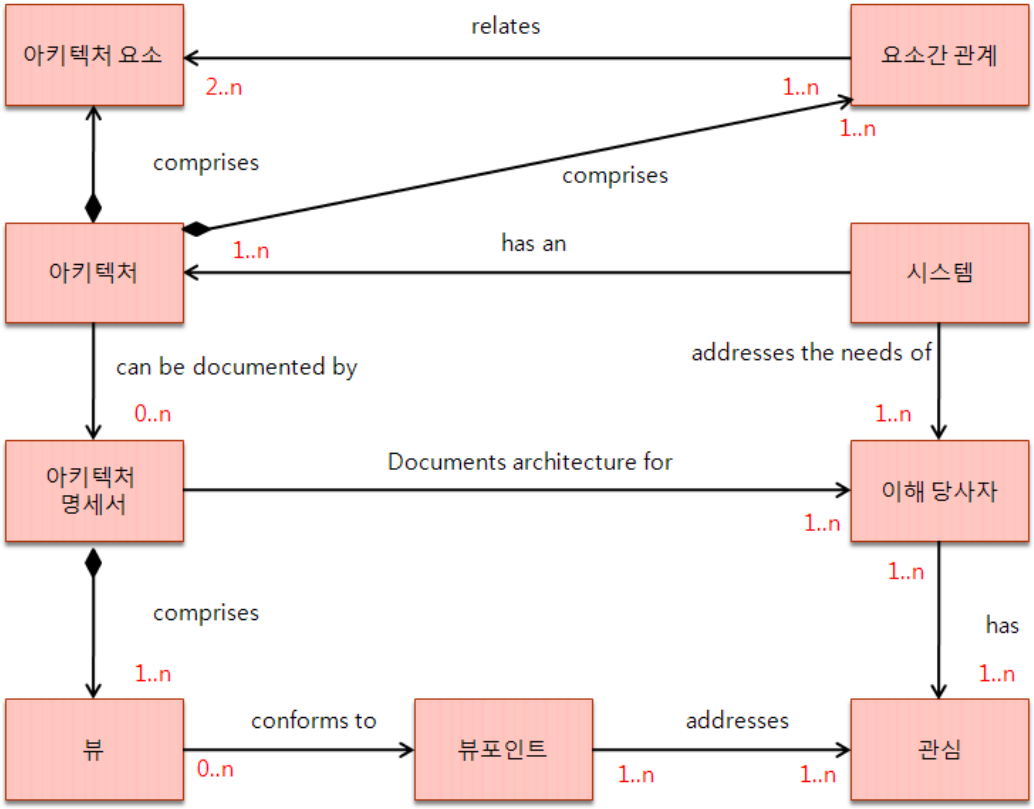
다. 문맥에서 뷰와 뷰포인트

그림 1은 다음과 같은 주요 요소 관계를 UML 클래스 다이어그램으로 표현하고 있다.

- 이해 당사자의 필요, 관심, 목표, 목적을 설명하기 위해 시스템을 구현한다.
- 시스템의 아키텍처는 아키텍처 요소와 관련 요소 간의 관계로 이루어져 있다.
- 시스템의 아키텍처는 아키텍처 명세서에 암시적으로 문서화된다.
- 아키텍처 명세서는 이해 당사자를 위한 아키텍처를 문서화하고 그들이 원하는 보여줄

수 있다.

- o 뷰포인트는 뷰 클래스의 목적, 의도된 의견, 내용을 정의하고 언급하고자 하는 뷰 클래스의 관심을 정의한다.
- o 뷰는 뷰포인트에 따라 정의하고, 관심의 해결을 전달한다.
- o 아키텍처 명세서는 여러 뷰로 이루어져 있다.



(그림 1) 문맥에서 뷰와 뷰포인트

라. 뷰포인트의 위험성

뷰와 뷰포인트를 사용하는 것이 소프트웨어 아키텍처 문제를 자동으로 해결해주지는 않는다. 비록 문제를 관리하기 위하여 뷰를 사용하여도 다음과 같은 위험이 발생할 수 있다.

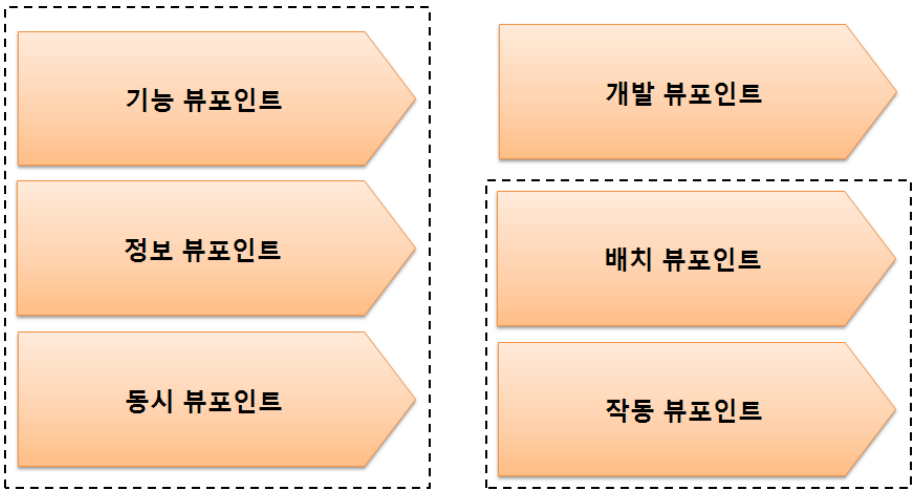
- o 불일치성: 여러 뷰를 사용함으로써 시스템은 필연적으로 일치성 문제를 야기한다.
- o 뷰의 잘못된 집합 선택: 아키텍처, 이해 당사자의 복잡성에 영향을 받는다.
- o 분열: 아키텍처 명세서를 따르기 어렵다.

마. 뷰포인트 카탈로그

정보 시스템 아키텍처를 위해 그림 2와 같은 6가지 주요 뷰포인트 카탈로그가 있다. 6가

지 뷰포인트는 세 가지로 그룹화가 가능하다.

- o 기능, 정보, 동시 뷰포인트는 시스템 기본 구조를 특성화한다.
- o 개발 뷰포인트는 시스템 구조를 지원하기 위해 존재한다.
- o 배치, 작동 뷰포인트는 실 환경에서 시스템을 특성화한다.



(그림 2) 뷰포인트 그룹화

제 2 절 소프트웨어 아키텍처 설계 프로세스

1. 개요

아키텍처를 설계하는 것은 비교적 어렵다. 아키텍처 설계는 프로젝트 생명 주기에서 범위와 요구 사항이 여전히 불명확한 이른 시점에 이루어진다. 그래서 현재 시스템의 뷰와 결과적으로 만들어질 것이 다를 수도 있다.

2. 지침 원칙

아키텍처 설계 프로세스에는 다음과 같은 지침 원칙이 있다. 첫째, 아키텍처 설계 프로세스는 이해 당사자의 관심에서 이루어져야 한다. 이해 당사자의 관심은 서로 충돌하거나 모순되는 의미를 내포할 수 있다. 그러므로 이런 관심의 균형을 맞추어야만 한다. 둘째, 아키텍처 의사 결정, 원칙, 해결 방법을 위해 효과적인 전달에 초점을 맞추어야 한다. 셋째, 생명 주기에 걸쳐 아키텍처 의사 결정과 원칙은 지켜져야 한다. 넷째, 하나 이상의 스탭이나 작업으로 구조화되어야 하고, 명확한 목적, 입력, 출력이 기술되어야 한다. 다섯째, 실용적이어야 한다. 여섯째, 특정 환경에 적용할 수 있도록 유연해야 한다. 일곱째, 관련 기술에 중

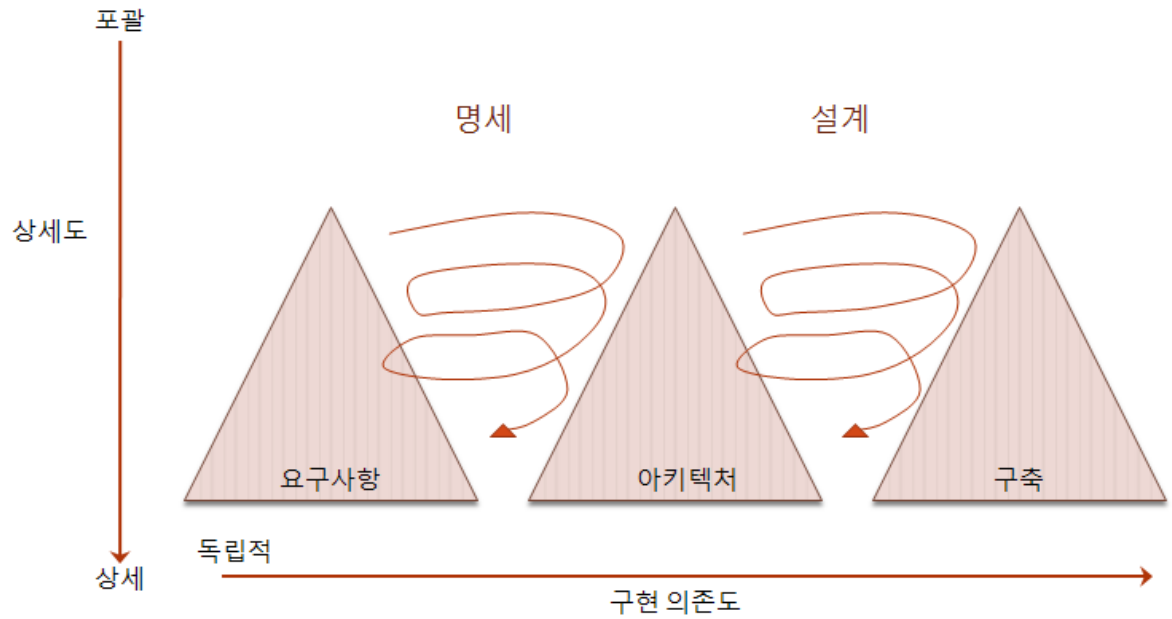
립적이어야 한다. 여덟째, 선택한 소프트에어 개발 생명주기와 통합할 수 있어야 한다. 아홉 번째, 좋은 소프트웨어 공학 실습과 ISO 9001과 같은 품질 관리 표준을 준수해야 한다.

3. 결과

아키텍처 설계 프로세스를 통해서 다음과 같은 결과가 나온다. 첫째, 요구사항의 설명과 다른 프로세스의 입력이 나온다. 둘째, 이해 당사자 기대를 관리한다. 셋째, 아키텍처 선택을 인식하고 평가한다. 넷째 아키텍처 인수 기준을 명세한다. 다섯째, 설계 입력의 집합을 생성한다.

4. 문맥

아키텍처 설계는 요구 사항 분석과 소프트웨어 구축 사이에 이루어 진다. 그림 3은 요구사항, 아키텍처, 구축 사이의 연동 관계를 잘 보여주는 Three Peaks 모델이다. 가운데 그려져 있는 화살표는 요구사항과 아키텍처, 아키텍처와 구현이 상세도가 증가하는 동안에 어떻게 합쳐지는지 보여준다.



(그림 3) Three Peaks 모델

요구사항은 시스템의 범위, 기능, 품질 특성을 정의함으로써 아키텍처 명세를 위한 문맥을 제공한다. 그리고 아키텍처는 요구사항을 좀더 명확하게 하거나 추가할 수 있도록 피드백을 제공하고, 이 요구사항의 우선 순위를 정해준다. 아키텍처는 계획된 시스템의 구축을 위한 입력 데이터를 구축에 제공한다. 그리고 구축은 아키텍처의 유효성과 잠재적인 문제에 대한 피드백을 제공한다.

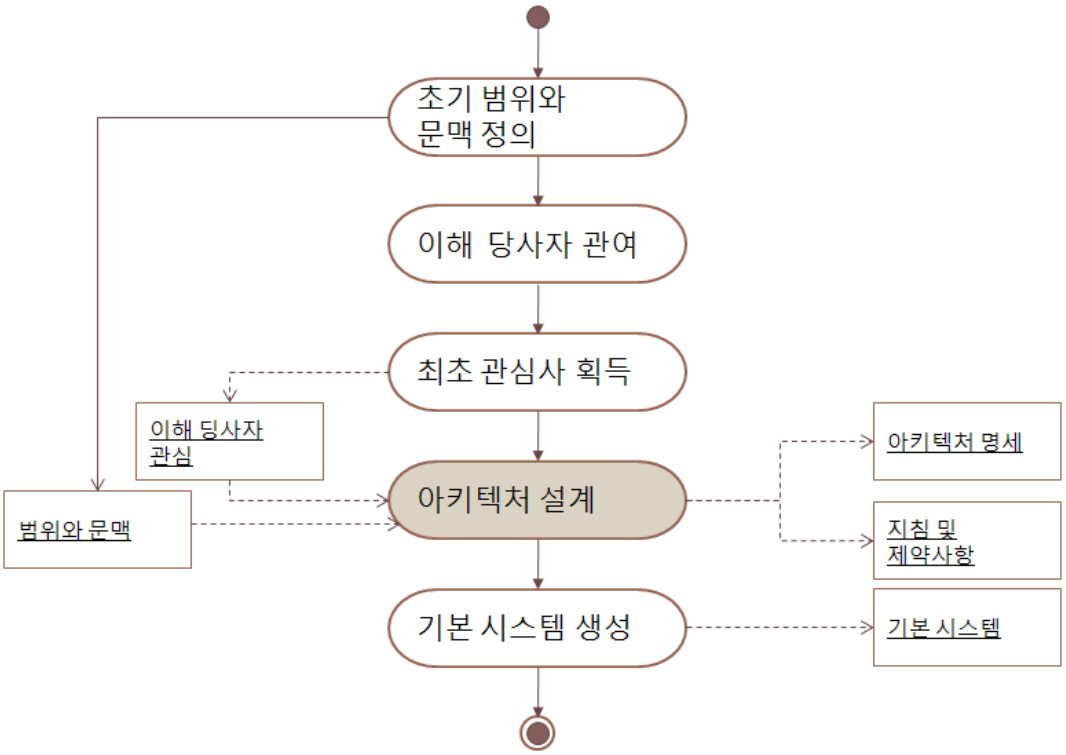
5. 종료 기준

아키텍처 명세 프로세스는 다음과 같은 상황에서 종료될 수 있다.

- o 아키텍처 명세를 정형적으로 검토하여 더 이상 중요한 의견이나 행동이 없을 때
- o 검토자가 아키텍처에 더 이상 중요한 이슈가 없다는 것에 동의하였을 때
- o 아키텍처 설계서가 사용자의 요구를 충분히 만족하였을 때, 단, 완벽한 버전을 요구하는 것은 아니다.

6. 지원 활동

그림 4는 아키텍처 설계시 지원할 수 있는 활동을 UML 활동 다이어그램으로 표현하고 있다.



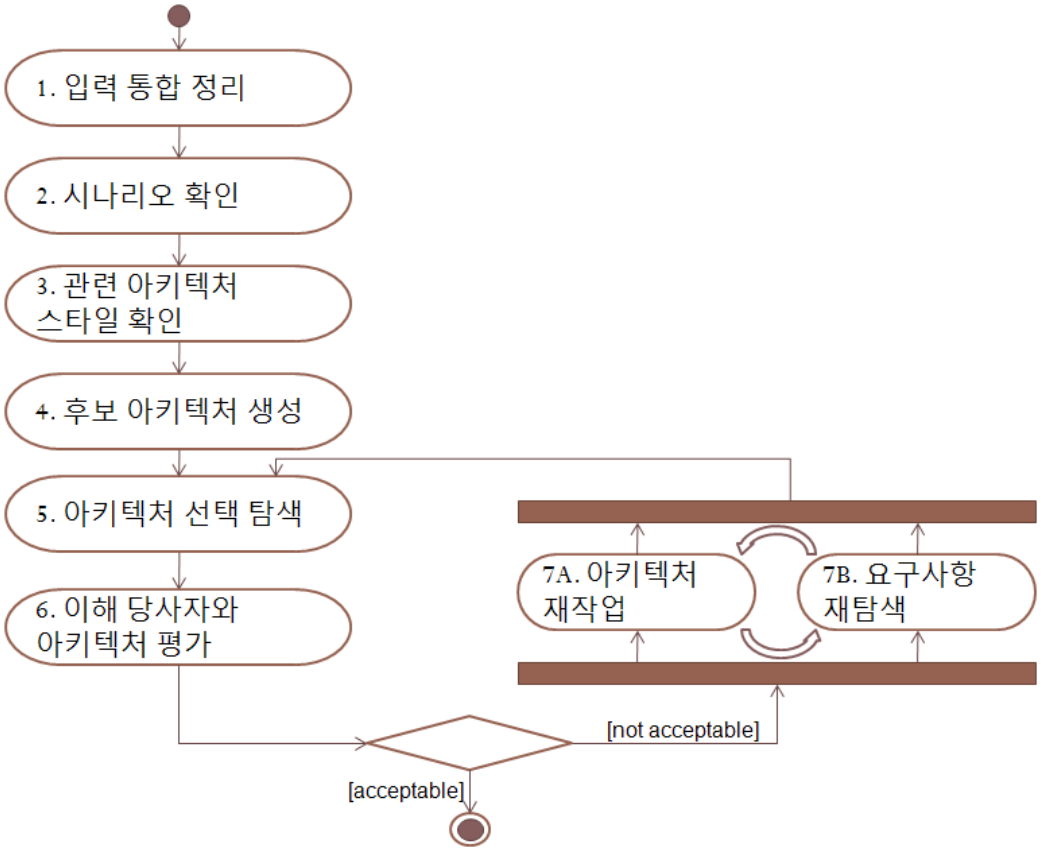
(그림 4) 아키텍처 설계 지원 활동

첫 번째 단계, 초기 범위와 문맥 정의에서는 시스템 행위, 역할의 범위를 정의하고 시스템이 존재하는 문맥에 대해 정의한다. 문맥은 작동과 조직화에 관련된 내용을 포함하고 있다. 입력으로는 필요, 비전 (vision), 조직에서의 전략, 전사적 IT 아키텍처가 필요하다. 출력으로는 시스템 목표에 관한 초기 기술, 역할에서 필요한 것과 불필요 한 것에 관한 기술,

초기 시스템 문맥 정의서가 나온다. 두 번째 단계, 이해 당사자 관여에서는 시스템의 중요한 이해 당사자를 인식하고 그들과 같이 일할 수 있게 관계를 만든다. 입력으로는 범위, 문맥, 조직화 구조가 필요하다. 출력으로는 이해 당사자 그룹의 정의, 하나 이상의 이름, 그룹을 대표하는 참여 사람이 나온다. 세 번째 단계, 최초 관심사 획득에서는 각 이해 당사자 그룹에서 시스템에 관해 갖고 있는 관심과 각 관심에 대한 우선 순위를 명확하게 이해한다. 입력으로는 이해 당사자 목록, 범위, 문맥이 필요하다. 출력으로는 이해 당사자가 우선시 하는 관심의 초기 정의가 나온다. 네 번째 단계, 아키텍처 설계에서는 시스템을 위한 아키텍처 명세서를 생성한다. 입력으로는 이해 당사자 목록과 범위 문맥이 필요하다. 출력으로는 아키텍처 명세서와 지침, 제약 사항이 나온다. 다섯 번째 단계, 기본 시스템 생성에서는 실행 가능한 아키텍처를 개발한다. 입력으로는 아키텍처 설계와 관련 지침과 제약사항이 필요하다. 출력으로는, 제한적으로 작동하는 시스템이 나오는데, 이 시스템은 적어도 하나 이상의 시나리오를 보여줘야 한다.

7. 설계 프로세스

그림 5는 아키텍처 설계 프로세스를 UML 활동 다이어그램으로 표현하고 있다.



(그림 5) 아키텍처 설계 상세 프로세스

첫 번째 단계, 입력 통합 정리에서는 초기 입력을 이해하고, 유효성을 확인하며, 정제한다. 입력으로는 범위, 문맥, 이해 당사자의 관심과 같은 가공되지 않은 프로세스 입력이 필요하다. 출력으로는 통합 정리된 입력이 나온다. 이를 위해서는 먼저 가공되지 않은 프로세스 인풋을 가져오고, 이들 간의 불일치성을 해결하여야 한다. 또한, 공개된 질문에 대해 답해야 하고, 필요로 한 부분은 깊게 조사한다.

두 번째 단계, 시나리오 확인에서는 시스템에서 가장 중요한 요구사항을 기술할 수 있는 시나리오 집합을 찾는다. 입력으로는 통합 정리된 입력이 필요하고 출력으로는 아키텍처 시나리오가 나온다. 이런 시나리오를 만들기 위해서는 두 가지를 고려해야 한다. 먼저, 시스템의 가장 중요한 속성을 특성화해야 한다. 또한, 시나리오를 통해 아키텍처를 평가할 수 있어야 한다.

세 번째 단계, 관련 아키텍처 스타일 확인에서는 하나 이상의 검증된 아키텍처 스타일을 찾는다. 입력으로는 통합 정리된 입력과 아키텍처 시나리오가 필요하고 출력으로는 아키텍처 스타일이 나온다. 이를 위해서는 아키텍처 스타일의 기존 카탈로그를 검토하고, 시스템이 작동하는 조직을 고려한다. 또한, 예상되는 아키텍처와 관련 있는 스타일을 찾는다.

네 번째 단계, 후보 아키텍처 생성에서는 기본 아키텍처 관심을 반영하고 있는 최초의 아키텍처를 개발한다. 입력으로는 통합 정리된 입력과 관련 아키텍처 스타일, 뷰포인트, 관점이 필요하고 출력으로는 아키텍처 뷰의 초안이 나온다. 이를 위해서는 초기 아키텍처 아이디어를 정의할 수 있는 아키텍처 뷰의 초기 집합을 만든다.

다섯 번째 단계, 아키텍처 선택 탐색에서는 다양한 아키텍처 가능성을 탐색하고, 그 중 하나를 선택하기 위하여 아키텍처 의사 결정을 한다. 입력으로는 통합 정리된 입력, 아키텍처 뷰의 초안, 아키텍처 시나리오, 뷰포인트, 관점이 필요하고 출력으로는 좀더 상세하고 정확한 아키텍처 뷰가 나온다. 이를 위해서는 시나리오를 모델의 초안에 적용하여 확인해 본다. 또한 위험, 관심, 불확실성과 관련된 분야를 고려해 본다. 그리고 하나 이상의 해결 방법이 있을 때는 각각의 장단점을 평가해 본다.

여섯 번째 단계, 이해 당사자와 아키텍처 평가에서는 주요 이해 당사자와 아키텍처 평가를 하고 문제나 결점을 확인하며, 이해 당사자의 인수를 받는다. 입력으로는 통합 정리된 입력과 아키텍처 뷰와 관점 결과가 필요하고 출력으로는 아키텍처 검토 해설이 나온다. 이를 위해서는 대표적인 이해 당사자 집단과 아키텍처를 평가해야 한다.

일곱 번째 단계는 아키텍처 재작업과 요구사항 재탐색이 반복적으로 일어날 수 있다. 먼저 아키텍처 재작업에서는 요구사항 재탐색에서는 평가 작업 동안 발생한 관심에 대해 언급한다. 입력으로는 아키텍처 뷰, 관련 아키텍처 스타일, 뷰포인트, 관점, 아키텍처 검토 해설이 필요하고, 출력으로는 재작업한 아키텍처 뷰, 좀더 조사해야 할 분야가 나온다. 이를 위해서는 아키텍처 평가의 결과를 가져오고, 좀더 나은 아키텍처를 만들기 위해 그 결과를 언급한다. 요구사항 재탐색에서는 시스템 본래의 요구사항 중 변경된 것을 고려한다. 입력으로는 아키텍처 뷰와 아키텍처 검토 해설이 필요하고 출력으로 개정된 요구사항이 나온다.

이를 위해서는 이해 당사자와 함께 요구사항을 다시 검토 하고 필요한 개정에 대해 동의를 얻어야 한다.

8. 소프트웨어 개발 생명 주기에서 아키텍처 설계

- o 폭포수 접근 방법: 생명 주기에서 분할된 하나의 작업으로 보고 요구사항 정의 이전, 이후에 수행될 수 있다.
- o 반복적 접근 방법: 분석 단계를 구성하거나 계속적으로 수행되는 활동이 될 수 있다.
- o 기민한 방법: 구축 부분에서 시작되어 반복을 통해 문맥이 맞춰지며 시스템에 걸쳐 일치성 및 기술적 무결성을 유지할 수 있다.

제 3 절 설계 지침

1. 범위, 고려사항, 지침, 그리고 제약사항

아키텍처 솔루션에는 비즈니스 목적들과 드라이버들, 아키텍처의 범위, 아키텍처에서 고려해야할 사항, 아키텍처의 지침 그리고 그 밖의 아키텍처의 제약사항들이 있다.

가. 비즈니스 목적과 드라이버

비즈니스 목적과 드라이버는 프로젝트를 위한 컨텍스트의 집합이고, 프로젝트가 존재하는 근본적인 이유가 된다. 또한 비즈니스의 목적은 기관들이 목표로 하는 것들을 명세한 것이다. 비즈니스 드라이버는 비즈니스를 보호하고 성장할 수 있도록 하기 위해서 특정한 방법으로 특정 기관에 영향을 미칠 수 있는 원동력이 된다.

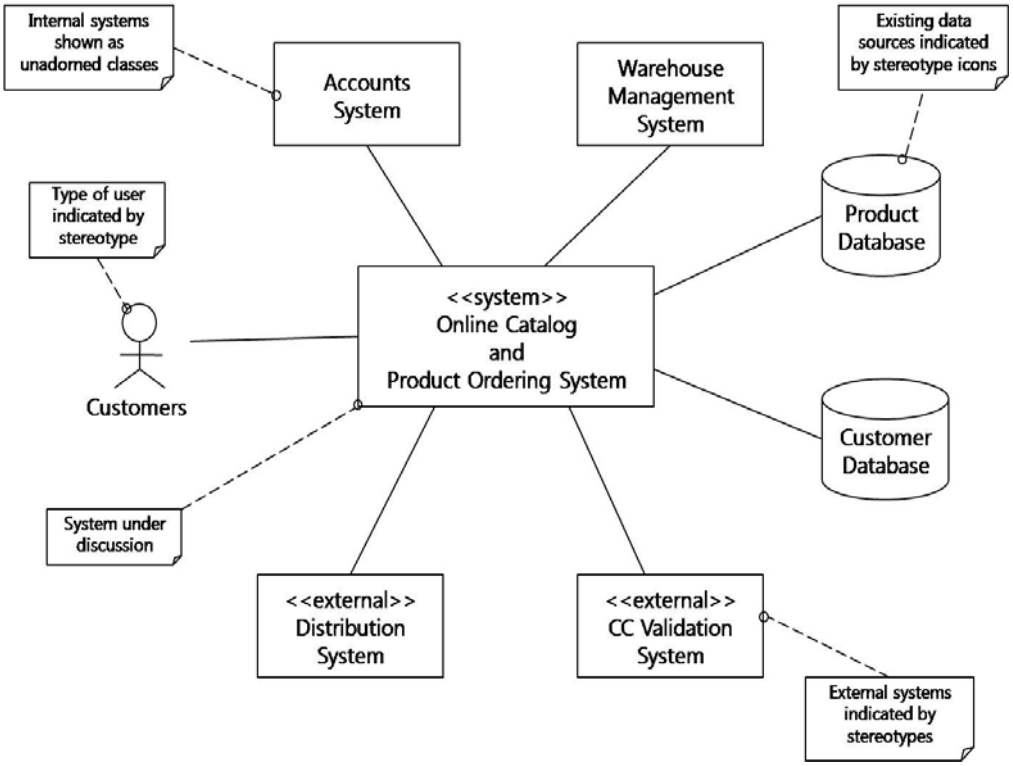
나. 아키텍처의 범위

- o 범위는 다음과 같이 정의된다.
  - 시스템으로부터 지원되는 기능들의 범위이다.
  - 시스템의 외부 인터페이스이다.
  - 새로운 시스템 안으로 어떠한 데이터든 입력할 수 있다.
- o 좋은 범위는 다음과 같이 정의 될 수 있다.
  - 정확하고 명확해야 하며, 간결해야 한다.
  - 자세한 수준에 적합할 수 있도록 지원되어야 한다.

- 상태가 명백해야 한다.
- 애매모호하거나, 특수용어, 전문용어 등은 회피해야 한다.
- 널리 보급되고 이해할 수 있어야 한다.

o 문맥 다이어그램

- 시스템의 경계는 높은 수준의 그림으로 표현된다. 그림 6은 문맥 다이어그램의 예를 보여주고 있다.



(그림 6) 컨텍스트 다이어그램의 예제

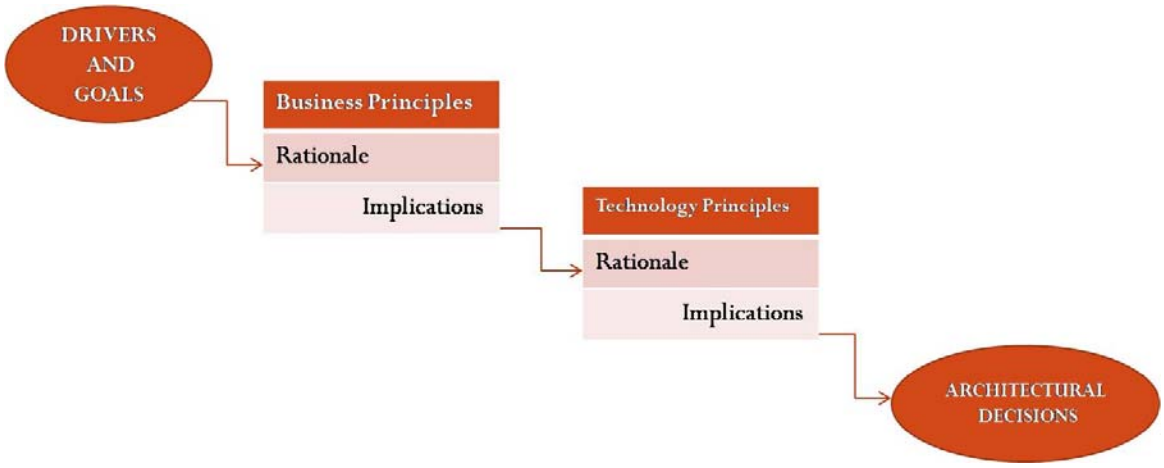
다. 고려 사항

아키텍처는 요구사항, 목적, 의도 혹은 이해당사자들의 목표들이 포함되어 있어야 한다. 또한 아키텍처는 다음과 같이 분류될 수 있다.

o 아키텍처의 목적들에는 목표 혹은 의도가 있다.
o 기능적 요구사항에는 시스템의 요구사항이 무엇인지 포함 되어 있어야 한다.
o 아키텍처 요구사항들에는 비기능적 요구사항도 있어야 한다.
o 아키텍처에서 중요하게 고려되어야 할 사항은 추적이 가능해야 하며, 정량화나 측정이 가능해야 하고 테스트가 가능해야 한다.

라. 지침사항

아키텍처 설계 지침사항에는 의도나 접근 방법, 신뢰할 수 있는 표현들이 기본적으로 들어가 있어야 한다. 아키텍처의 좋은 지침은 발전성이 있어야 하며, 비즈니스 드라이버나 목적 그리고 다른 지침으로부터 강력한 동기가 있어야 한다. 명확하게 잘 정의 될 수 있도록 해야 하며, 테스트에 대한 지침과 중요한 항목들에 대한 지침이 포함 되어 있어야 한다. 그림 7은 지침을 사용하여 추적할 수 있는 검증 방법이다.



(그림 7) 지침을 이용한 추적 검증

2. 이해당사자 식별

가. 다른 아키텍처의 제약사항

- 공개된 표준(Open Standards)을 사용해야 한다. 즉, ISO(International Standardization Organization, 국제 표준화 기구)와 같은 형태로 맞추고 정의 되어야 한다.
- 마이크로 소프트(Microsoft), 선(SUN), 오라클(Oracle)와 같은 회사로부터 관리되고 생성되는 소유권이 있는 표준을 사용한다.
- 기관이나 기구에서 사용하기 위해 개발된 기구 표준을 사용한다.
- 지침사항이나 전략, 방침이 있어야 한다.

나. 이해당사자 선택

이해당사자 선택은 주관적인 활동이다. 그러나 이해당사자는 제품이나 시스템개발의 성공만을 목적으로 한 사람들이 아니라, 일반적이고 도메인에 대해 넓은 지식을 가진 사람들이어야 한다. 아키텍처에서 좋은 이해당사자들은 다음과 같다.

- 이해당사자는 견문이 넓고 지식이 많은 사람이어야 한다.
- 이해당사자는 자기의 입장이나 의견을 명확하게 말할 수 있는 사람이어야 한다.
- 이해당사자는 정당하다고 인정이 되면 다른 사람에게 권한을 부여 할 수 있어야 한다.
- 이해당사자는 대표적인 사람이어야 한다.

다. 이해당사자들의 역할들

- 획득자(Acquirer): 시스템이나 혹은 제품의 획득을 감시하는 사람이다.
- 평가자(Assessor): 표준이나 합법적인 규정을 기반으로 하여 시스템과 일치하는지 감시한다.
- 개발자(Developers): 명세서를 기반으로하여 시스템을 조립하거나 배치한다.
- 유지보수자(Maintainers): 시스템을 잘 운영하기 위해서 시스템 평가와 관리를 한다.
- 공급자(Support Staff): 운영되고 있는 시스템이나 제품을 사용하는 사용자들을 지원한다.
- 시스템 관리자(System Administrators): 운영되고 있는 시스템을 잘 배치시키는 작업을 한다.
- 시험관(Testers): 시스템이 사용하기에 적합한지 검증하기 위해 시험한다.
- 사용자(Users): 시스템의 기능성이나 시스템을 사용하는데 적적할 시기를 정의한다.

라. 이해당사자들의 책임

- 이해당사자들은 고려해야 할 모든 사항들에 대해서 정확한 커뮤니케이션 내용을 아키텍처 설계자들에게 전달해야 한다.
- 대표자들이나 대리 이해당사자들은 사람들이 나타내고자 하는 모든 고려사항들을 아키텍처 설계자들에게 정확하게 전달해야 한다.
- 이해당사자들은 적절한 시기에 결정할 수 있도록 해야 하고 정당한 방법을 사용해야 한다.
- 만약 이해당사자들이 결정 권한을 가지지 못한다면, 단계적으로 적합하고 누군가 힘이 있는 사람으로부터 결정권을 획득한다.

3. 시나리오 확인과 사용

가. 아키텍처 시나리오

- 아키텍처 시나리오는 시스템이 필요로 하는 것에 대해 응답할 수 있는 정의들과 생산

환경이 적합한 시스템들에 대한 상황들에 대해서 명확하고 간결하게 표현한 것이다.

o 이러한 시나리오를 통해서 유저와의 대화, 특정한 상황에서 일어나지 않으면 안 되는 처리, 일어날 수 있는 특정 최대 부하 상황, 시스템에서 발생될 수 있는 변화 그리고 시스템에서 필요 하는 다른 어떤 특정 상황들을 알 수 있다.

o 시나리오들의 타입

- 기능적 시나리오들
  · 기능적 시나리오들은 외부 이벤트들의 연속적인 관점에서 항상 정의되어야 한다.
- 시스템 품질(비기능적) 시나리오들
  · 시스템 품질 시나리오들은 하나 이상의 고유한 품질특성을 설명하기 위해 그 환경들에서 시스템이 어떻게 작용되고 변경되는지 그 관점에서 정의되어야 한다.
- 시나리오들을 위한 사용
  · 아키텍처를 정의하기 위한 입력: 창의력이나 아이디어들은 많은 공간에서부터 올 수 있다.
  · 아키텍처 평가: 시나리오들은 아키텍처 평가의 거의 대부분의 프로세스 내부에서 중요하게 작용된다.
  · 이해당사자들과의 커뮤니케이션: 시나리오의 논의와 시스템들의 상황표현에 대해서 이해당사자들의 모든 부류의 사람들과 함께 커뮤니케이션을 통하여 이루어질 수 있으며, 이는 매우 유용한 수단이다.
  · 못 찾은 요구사항 찾기: 시나리오 생성의 다른 장점은 종종 못 찾았던 요구사항 뿐만 아니라 이미 알맞게 존재했던 것들을 재사용 할 수 있다. 모든 것들은 시나리오를 통하여 찾을 수 있다.
  · 테스스팅 프로세스 활동: 시나리오들은 중요한 이해당사자들의 생각들을 강조하기 위해서 많은 도움을 준다. 앞서 말한 바와 같이 시나리오들은 테스트 활동의 초점을 위해 아주 유용한 가이드라인을 제공한다.

나. 식별과 시나리오의 우선순위

o 시나리오들을 식별하기 위한 출처

- 요구사항들은 기능적 요구사항들과 시스템 품질 요구사항 즉, 비기능적 요구사항들을 포함한다.
- 이해당사자들은 매우 유용한 시나리오들을 가지고 있다.
- 경험을 위해 대용할 수 있는 시나리오는 없다. 시나리오들 중에서 가장 가치 있는 것은 경험이다.

o 시나리오 우선순위를 위한 두 개의 기준

- 이해당사자들의 관찰지점으로부터 중요한 경험이 나온다.

- 아키텍처 설계자들의 관찰지점으로부터 위험성이 나온다.

o 지침사항

- 이해당사자들은 시나리오 우선순위에 대해 연루될 수 있다.
- 이해당사자들의 우선순위가 변경될 때, 아키텍처 설계자는 다른 이해당사자들의 관점에 대해서 결정권이 필요하다.
- 아키텍처 설계자는 다른 어떤 시나리오들을 위탁하지 않고, 어떤 문제들에 대해서 고려한다.
- 시나리오 계층은 우선순위와 위험사이에서 결정된다.

다. 시나리오 획득

o 기능적 시나리오들

- 개요: 시나리오 장점은 예제를 설명하는데 있다.
- 시스템상태: 시나리오에는 시스템의 상태가 표현되어 있다.
- 시스템 환경: 환경에 대한 중요한 관찰은 운영되는 시스템이다. 예를 들면, 외부시스템 사용 불가능, 시간제약 등이 있다.
- 외부자극: 시나리오에는 원인에 대한 정의가 표현되어 있다.
- 요구되는 시스템 응답: 시스템에 대한 설명은 시나리오의 응답이나 외부 관찰자들의 관찰 지점이다.

o 시스템 품질 시나리오들

- 개요: 시나리오 장점은 예제를 설명하는데 있다.
- 시스템 환경: 환경에 대한 중요한 관찰은 운영되는 시스템이다. 예를 들면, 외부시스템 사용 불가능, 시간제약 등이 있다.
- 환경 변경: 사나리오에는 시스템 환경의 변경에 대한 설명이 되어 있다. 예를 들면, 외부 시스템 행위의 변경, 보안공격, 요구사항 수정 등이 있다.
- 요구되는 시스템 행위: 시나리오에는 시스템의 환경이 변경되어 응답되는 행위에 대해서 반드시 정의되어 있다.

라. 시나리오 적용

o 페이퍼 모델(Paper Model)은 단순하거나 대부분의 공통적인 시나리오에서 시스템의 응답을 보여주기 위해 적용된 시나리오이다. 즉, UML이나 데이터흐름이다.

o 워크스루(Walkthrough)는 페이퍼 기반의 모델을 증명하는데 사용된다.

o 시뮬레이션은 컴퓨터 기반 시뮬레이션을 개발하는 가이드이다.

o 테스트 구현 프로토타입: 프로토타입 가이드는 이해당사자들의 중요한 특성의 높은 위

험 범위에 효과적인 프로토타입에 초점을 맞춘다.

o 최대규모 실시간 테스팅: 실시간 시스템레벨 테스트 계획을 위한 기반이다.

4. 스타일과 패턴 사용

소프트웨어 패턴의 목적은 쉽게 재사용할 수 있고, 표준 폼에서 특정한 디자인 문제에 대해 검증하고 광범위한 어플리케이션 솔루션들을 공유하기 위해서이다.

o 정보표현
- 이름: 의미 있는 이름은 패턴을 논의하거나 명확하게 식별하기 위해서는 가능하다.
- 문맥: 패턴을 위한 전력이나 상황들을 표현한 것이다.
- 문제: 문제의 진술은 패턴을 사용하여 해답을 주거나 효과적으로 적용하기 위해 필요한 조건을 말한다.
- 솔루션: 문제 해결의 표현은 디자인 모델 폼에 패턴을 적용한다.

o 스타일, 패턴, 이디엄
- 아키텍처 스타일은 소프트웨어 시스템을 위해 기관 스키마로 기본 구조를 표현하고, 요소타입을 미리정하는 집합들을 지원하고, 규정과 가이드라인으로부터의 책임을 명세한다. 또한 아키텍처 스타일은 시스템 수준 기관을 위한 솔루션이며, 아키텍처 스타일 구조는 아키텍처 요소의 타입과 인터페이스 그리고 연결장치 타입, 마지막으로 요소와 연결장치들의 구조는 합쳐진다.
- 디자인 패턴은 디자인 패턴사이의 관계 혹은 소프트웨어 시스템의 요소들을 정제하기 위한 스키마를 지원한다. 연결되는 디자인 요소의 공통적으로 순환되는 구조를 표현한다. 또한 디자인 패턴은 자세한 소프트웨어 디자인 문제에 대한 해결책을 가지고 있다.
- 이디엄은 프로그래밍 언어를 명세하기 위한 낮은 수준의 언어이다.
- 스타일, 패턴, 이디엄의 도움이 되는 규칙
  · 특정한 도메인에서 문제의 특정 타입을 해결하기 위한 지식 상점이다.
  · 실행 가능한 좋은 디자인의 예제가 있다.
  · 논의되는 디자인 문제들을 위한 공통적인 언어이다.
  · 표준들로 구성되어 있다.

5. 아키텍처 모델 생산

가. 모델의 정의

아키텍처 관점의 추상적이거나 단순화된 표현이고 모델의 목적은 하나 이상의 이해당사

자들에게 시스템 관점을 전달한다.

나. 주요 요소들 간의 관계

- o 아키텍처 명세서에 아키텍처는 기술되어 있다.
- o 아키텍처 명세서는 아키텍처의 여러 뷰로 구성되어 있다.
- o 각 뷰의 내용은 뷰포인트에 기반을 둔다.
- o 관점은 여러 모델로 구성되어 있다.
  - 모델은 관점과 관계된 아키텍처의 독특한 퓨처를 표현하는 방식이다.
  - 모델은 프로세스에서 중추적인 역할을 한다.
- o 관점을 적용하면 모델이 변화 할 수 있다.
  - 보안 관점

다. 모델의 중요성

- o 모델은 모델링하는 상황을 이해하는데 도움을 준다.
- o 모델은 우리 생각을 다른 사람들에게 이해하도록 도와주는 통신 매개체 역할을 한다.
- o 모델은 상호관계를 이해하고 주요 요소를 분리하는데 도움이 되는 상황을 분석하는데 도움이 된다.
- o 모델은 프로세스, 팀, 및 산출물을 구성하는데 도움이 된다.

라. 모델 형태

(1) 정성적 모델

모델화 되는 아키텍처의 주요 구조적이거나 행위적인 요소, 퓨처 및 매개 변수를 표현한다. 건축물의 모델과 청사진을 설계하는 것과 비슷하다. 아키텍처 명세서의 뷰의 주요한 내용을 형성하고 어떤 관점은 산출물이 중요하다. 정성적 모델의 형태는 기능적 구조 모델 또는 정보 모델과 같은 도식화된 아키텍처 모델 형태다.

(2) 정량적 모델

성능, 회복 및 수용력과 같은 아키텍처의 측정 가능한 특성에 관한 문서를 가지고 있다. 건축물을 만드는데 필요한 수학적 모델과 비슷하다.

예. 활용도 = 업무 처리량 X 업무당 작업 시간

(3) 개요

개요는 비전문가에게 아키텍처의 중요한 관점을 설명하기 위하여 만들고 정규화 되지 않는 그래픽 모델이다. 여러 모델링 표시, 그림 및 아이콘 요소로 구성되어 있다. 새로운 건축물의 영향을 사람들이 이해하도록 만들어진 예술가의 인쇄물과 유사하다. 애매모호하게 만들고 혼란과 오해를 일으킬 위험이 있다.

마. 모델링 언어

(1) 아키텍처 명세 언어 (ADL)

컴퓨터 시스템에서 아키텍처를 정의하는데 모델을 위한 특수 목적의 표기법이다. 아키텍처 수준을 상세하게 만들기 위하여 설계할 수 있는 장점을 가지고 있다. 아키텍처 명세 언어는 Aesop, Unicon, 및 xADL 등이 있다.

(2) Unified Modeling Language (UML)

소프트웨어 집약 시스템의 결과물을 시각화, 명세화, 구조화, 및 문서화하기 위한 도식화된 언어이다. 널리 사용되고 유용하고 확장 가능성을 가지는 표기법을 포함하고 있다. UML를 아키텍처 명세서 작성하는데 사용하고 아키텍처 명세 언어로써 사용하여 UML 사용 능력을 향상 시킨다.

(3) 다른 모델링 언어

데이터 모델링하기 위한 엔티티 관계 모델과 같은 특정 도메인 모델링 언어가 필요하다.

바. 효율적인 모델을 위한 지침서

(1) 목적에 맞게 설계하기

모델을 하기 위한 확실한 목적이 없으면 모델을 표현하는 상세화 수준, 완벽도, 형식이 확실하지 않는다.

(2) 모델 관계자에게 집중하기

여러 관계자는 같은 모델을 표현하는데 여러 형태로 표현하기를 원한다.

(3) 추상화에 주의하기

추상화는 뚜렷하지 않거나 모호한 의미가 아니다. 추상화 모델은 실제 모델보다 명확하고 정확해야 한다.

(4) 명세 이름 선택하기

요소 이름은 효율적인 대화하는데 중요한 영향을 준다. 거추장스러운 단어보다는 명확한 단어를 선택하는 게 중요하다.

(5) 용어 정의하기

각기 다른 해석을 하지 않고 명확하게 실세계와 연결되는 의미와 역할을 하는 요소를 정의하는데 충분한 시간을 투자한다.

(6) 단순화 지향하기

모델 복잡도가 증가하면 대화하고 분석을 위한 효율성이 급속도로 줄어든다.

(7) 정의된 표기법 사용하기

모델의 내용에 이해당사자들이 의미를 의심하지 않고 내용에 집중할 수 있도록 표기법 정의를 확실하게 해야 한다.

(8) 모델 검증하기

모델은 사실을 추정하기 때문에 추정의 타당성을 확인하는 게 중요하다.

(9) 모델의 활동성 유지하기

모든 것은 소프트웨어 프로젝트 개발 기간 동안 바뀐다. 요구사항은 나타나고 사라지고 새로운 제약사항이 나타나고 우선순위도 변화한다.

사. 에자일(agile) 모델링 기법

대부분 시점에서 확장과 도출 가능한 모델링하는 접근 방식이다. 모델은 끊임없이 변화하는 현실을 추정하기 때문에 좋은 모델 개발을 촉진하여 인식하는데 기반을 둔다.

6. 아키텍처 명세 생성

아키텍처 명세서는 이해당사자들의 관심에 부합하는 아키텍처를 이해하고 확실하게 하는 방법으로 아키텍처를 문서화하는 결과물의 집합이다.

가. 효율적인 아키텍처 명세서 특성

(1) 정확성

o 정확성 두 가지 기준
- 이해당사자의 요구와 관심을 명확하게 표현해야 한다.
- 필요에 부합한 아키텍처를 명확하게 정의해야 한다.
  · 선택된 아키텍처는 이해당사자 요구와 부합하는가?
  · 명확하게 문서화 했는가?

(2) 효율성

아키텍처에 관하여 중요한 탐구를 나타내기 위하여 상세한 정보를 포함해야 한다. 효율성을 확인하는 좋은 방법은 적합한 뷰포인트와 뷰를 선택하고 적합한 관점을 적용해야 한다.

(3) 간결성

아키텍처적으로 중요한 것에 초점을 맞춘다. 이해 당사자의 능력과 경험, 새롭거나 친숙하지 않는 기술이 필요한 정도, 해결하고자 하는 문제의 어려운 정도, 아키텍처 명세서를 만들고 획득하기 위하여 가능한 시간과 자원에 따라 상세화 정도를 결정한다.

(4) 투명성

아키텍처 명세서에서 얻을 수 있는 최고의 품질은 모든 이해당사자들이 이해하도록 하는 것이다.

(5) 전달성

아키텍처는 시간이 지나면서 진화한다. 아키텍처 변화를 아키텍처 명세서에 반드시 반영을 해야 한다.

(6) 명확성

아키텍처 명세서는 시스템을 설계하고 구현하는데 있어서 충분히 자세하고 명확하게 시스템의 아키텍처 구조를 설명해야 한다.

나. 용어

독자들이 명확하게 이해하지 못한 용어는 아키텍처 명세서에 용어집을 추가한다. 조직이나 기업에서 사용하는 표준화된 정의에 기반을 하여 작성한다.

다. IEEE 표준

아키텍처 명세서를 하기위한 IEEE 1471 Recommended Practice는 소프트웨어 시스템 아키텍처 지침을 포함한 정형화된 표준이다.

o 아키텍처를 문서화하는 여섯 가지 추천하는 지침
- 아키텍처 문서 : 아키텍처 명세서는 표준 컨트롤과 컨텍스트 정보를 포함한다.
- 이해당사자와 관심 인식 : 아키텍처 명세서는 이해당사자와 그들이 원하는 관심을 인식해야한다.
- 아키텍처 뷰포인트 선택 : 아키텍처 명세서는 뷰포인트를 인지하고 선택한 뷰포인트에 관한 논리적 근거를 설명하고 뷰포인트가 각 관심과 연관성을 설명해야 한다.
- 아키텍처 뷰 : 아키텍처 명세서에 하나 이상의 뷰가 존재하고 이러한 뷰는 여러개의 모델로 구성되어 있고 관련된 뷰포인트와 연관성을 갖는다.
- 아키텍처 뷰와의 일관성 : 아키텍처 명세서는 여러 뷰를 고려하여 일관성 있게 분석하고 알려진 불일치를 기록해야 한다.
- 아키텍처 논리적 근거 : 아키텍처 명세서는 아키텍처를 구성할 때 선택기준에 대한 논리적 근거를 포함하고 충분히 검토된 선택 사항을 표시해야 한다.

라. 아키텍처 명세서 내용

(1) 문서 컨트롤

아키텍처 명세서의 버전을 인식하는데 필요하다.

(2) 목차

워드 프로세스 프로그램에서 제공하는 목차를 사용할 수 있다.

(3) 도입과 관리 개요

- o 아키텍처 명세서를 작성하는 목적 설명
- o 서술된 시스템 목표 요약
- o 범위와 주요 요구사항 요약
- o 해결책에 관한 상위 수준 개요 표시
- o 해결책의 이점, 구현상의 위험 및 전략 완화를 강조

(4) 범위 정의

- o 시스템이 제공하는 폭 넓은 기능적 범위
- o 시스템의 외부 인터페이스와 외부 인터페이스를 통하여 연결된 외부 시스템
- o 시스템이 사라지거나 수정
- o 새로운 시스템에 정보가 통합

(5) 요구사항과 관심 개요

- o 목표 : 시스템의 비즈니스와 기술적 목표를 표현한다.
- o 기능적 요구사항 : 시스템이 해야 하는 일을 정의한 요구사항 리스트이다.
- o 필요한 품질 특성 : 직접적으로 기능성을 요구하지 않고 시스템이 작동하는 방법을 설명한 요구사항 리스트이다.

(6) 일반적 아키텍처 원리

각 원리는 열거되고 논리적 근거와 구현을 포함해야 한다.

(7) 뷰

- o 기능적 뷰 내용
- o 특정 뷰 아키텍처 원리
- o 뷰 모델
- o 특성 향상
- o 시나리오 어플리케이션
- o 주석

(8) 품질 특성 요약

필요한 품질 특성을 준수하는 시스템 능력을 좀더 이해하기 편하게 일반적인 식견을 제공한다.

(9) 중요한 시나리오

초기 시스템 상태와 환경, 외부 자극물, 및 시스템에 필요한 행동을 기술한다.

(10) 부록

- o 참조문헌 또는 정보 출처
- o 용어나 단축 해설
- o 이해당사자 지도
- o 범위, 기능적 요구사항, 및 품질 특성에 관한 상세한 명세서
- o 요구사항과 아키텍처 퓨처에 관한 지도
- o 사용자가 사용하는 아키텍처 스타일, 디자인 패턴에 관한 설명
- o 상세 뷰 모델
- o 상세 특성 모델과 관찰
- o 어플리케이션 시나리오 상세 정보
- o 정책, 표준, 및 지침
- o 아키텍처 명세서 정형적 리뷰 결과물
- o 추가적인 지원 문서

7. 아키텍처 유효성 검증

가. 아키텍처 평가 이유

o 아키텍처 평가는 아키텍처 명세서가 한계가 있기 때문에 가치가 있다.
- 추상화 검증 : 아키텍처 명세서는 실세계를 추상화 한 것이다.
- 기술적 정확성 검사 : 아키텍처 명세서는 정적이고 컴퓨터가 직접적으로 실행 가능하지 않다.

o 평가는 대화적 관점에서 유용한 프로세스이다.
- 아키텍처 판매 : 아키텍처 평가 프로세스는 이해당사자의 요구에 합당한지 보여줌으로써 아키텍처 판매를 도와준다.
- 아키텍처 설명 : 상호 아키텍처 평가 프로세스는 비전문적 시스템 이해당사자들에게 아키텍처의 주요한 특성을 설명하는 효율적인 방법이다.

o 소프트웨어 개발 프로세스에서 아키텍처 평가를 하는 것이 좋다.
- 가정 평가
- 관리 결정 포인트 제공
- 공식 합의 기초를 제공
- 기술적 통합 확인

나. 평가 기술

o 프레젠테이션
o 공식 검토 및 조직적 워크스루
o 시나리오 기반 측정
o 프로토타입

(1) 프레젠테이션

단순한 형태의 아키텍처 평가는 이해당사자에게 제안된 아키텍처를 비공식적으로 설명한다.

(가) 장점

o 빠르게 만들 수 있고 여러 청중들에게 쉽게 적용시킬 수 있다.

- o 참석자들이 지참해야하는 준비물이 거의 필요 없고 노력이 거의 들지 않으며 쉽게 이해할 수 있다.
- o 참석자들의 반응과 질문에 즉각적으로 피드백을 줄 수 있다.

(나) 한계점

- o 프레젠테이션 미팅 동안에 피상적인 수준으로 분석된 결과물을 얻는다.
- o 이 접근법의 효율성은 프레젠테이션 자료에 의존적이다.
- o 참석자 준비 부족은 중요한 식견을 잊고 시간 부족으로 인하여 아키텍처에 안 좋은 영향을 준다.

(2) 공식 검토 및 조직적 워크스루

- o 공식 검토는 이해당사자와 아키텍처 명세서를 평가하는데 효율적인 방법이다. 참석자의 역할은 조정자, 발표자, 검토자로 구성된다.
- o 조직적 워크스루는 여러 사람이 명세서, 설계, 및 코드가 요구사항과 맞는지 상세하게 평가한다.

(가) 장점

프레젠테이션보다 참여자가 더 심도 있게 참여를 하고 좀 더 가치 있는 통찰을 할 수 있다.

(나) 한계점

- o 프레젠테이션보다 상대적으로 참여자들이 준비해야 하는 비용으로 인하여 좀 더 비싸다.
- o 모이기 전에 준비하는 품질에 따라서 결과가 결정된다.

(3) 시나리오 기반 측정

시나리오 기반 아키텍처 평가는 아키텍처가 이해당사자 요구와 얼마나 잘 부합되는지 평가하는 조직적인 접근이다.

- o 시나리오 기반 평가 방법

- Architecture Tradeoff Analysis Method (ATAM)
- Software Architecture Assessment Method (SAAM)

o 시나리오 기반 접근법의 다섯 가지 기본 단계

- 요구사항 이해 : 이해당사자들이 잘 이해하는지 확인하기 위하여 시스템 요구사항을 검토한다.
- 제안된 아키텍처 이해 : 평가자는 제안된 아키텍처를 상사하게 이해한다.
- 중요한 시나리오 인식 : 특정 시나리오에 아키텍처가 부합한지 고려하는 작업이 필요하다.
- 아키텍처 분석 : 특정 상황에서 아키텍처가 요구와 얼마나 잘 부합되는지 분석하고 각 시나리오에 시나리오 기반 기술이 고려되어야 한다.
- 결론 도출 : 평가자는 아키텍처 적합성에 관한 특별한 결론을 기술하고 보고서에 실행 결과를 작성한다.

(가) 장점

특정 아키텍처 접근법의 장단점에 관한 심도 있고 면밀한 분석을 제공한다. 아키텍처가 생성하는 절충점(tradeoff)에 관하여 명확하게 이해하게 하고 이해당사자들에게 논리적 근거를 설명하는데 도움이 된다. 아키텍처 팀이 만든 결정을 이해하고 이러한 결정을 하게 된 이유와 연관성을 알 수 있다.

(나) 한계점

단순한 검토나 워크스루에 비하여 적용하기에 너무 복잡하고 비용이 많이 소요된다. 교육이나 중요하게 준비하는 게 필요하다. 이해당사자가 적극적으로 참석하지 않으면 프로세스는 이득이 적을 수 있다.

(4) 프로토타입과 기본적인 개념 증명 시스템

기술적 위험을 줄이고 사용자 인터페이스를 설계하는데 도움이 된다.

프로토타입은 피드백이나 평가를 위하여 사용자에게 보여주는 시스템의 기능적 부분집합이다.

기본적인 개념 증명은 제한된 아키텍처의 위험 요소를 증명하기 위해 설계된 코드가 실현가능성이 있는지 증명하기 하고 문제점이나 함정을 강조하기 위한 방법이다.

(가) 장점

쉽게 변화하는 생명주기의 특정 포인트에서 기술적 결정을 구체적인 평가를 제공한다. 시스템을 구현하기 전에 안전한 환경에서 시스템 구현 기술에 관하여 배우고 고려하는 기회를 제공한다. 운영중 프로세스, 기술, 및 이해당사자에게 신뢰감을 증가시키기 위하여 실례를 보여준다.

(나) 한계점

생성하는데 시간과 비용이 많이 들기 때문에 중요한 결정에서만 이용한다.

(5) 근간이 되는 시스템

시스템 구조를 구현한 초기 버전의 시스템은 최소의 시스템 기능성을 포함한다.

(가) 장점

근간이 되는 시스템은 가능한 아키텍처 평가를 철저하게 하고 설득력 있게 한다. 평가 활동을 통하여 실제적으로 전달할 수 있다.

(나) 한계점

아키텍처 평가에서 최고로 비용이 많이 들어가는 형태이고 근간이 되는 시스템을 만들기 위하여 많은 개발 경험과 소프트웨어 공학 학습이 필요하다.

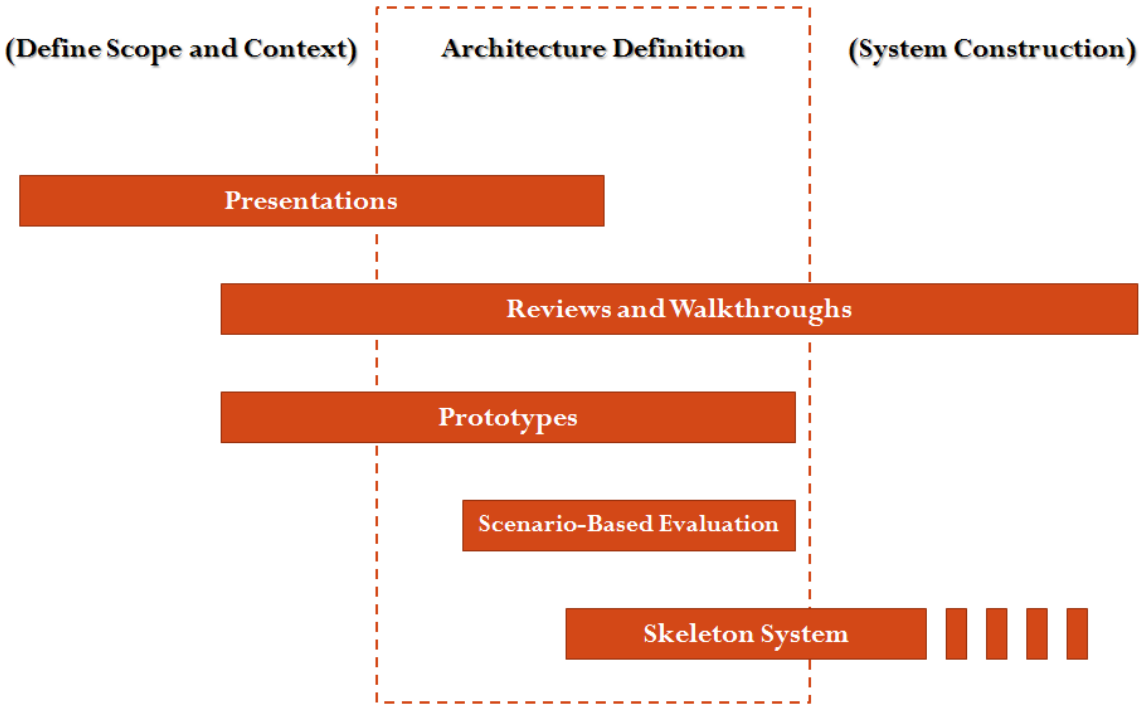
(6) 시나리오 기반 평가 방법

시나리오 기반 평가 방법은 시스템 이해당사자에게 중요한 시스템 사용 시나리오와 시스템 특성을 평가하는데 도움이 된다.

- o 시나리오 기반 평가 방법
  - SAAM : 기능적 시나리오를 가지고 본질적이고 단순한 메소드를 평가한다.
  - ATAM : 품질 특성 시나리오를 이용하여 차후에 정교하게 개발하는 접근방법이다.

다. 소프트웨어 생명 주기 동안 평가

평가는 소프트웨어 개발 생명 주기 중 특정 시점에서 수행하는 단일한 활동으로 생각하면 안 된다. 소프트웨어를 개발하는 동안에 시스템 아키텍처를 주기적으로 평가하는 지속적인 작업으로 간주되어 진다.



(그림 8) 생명주기에서 여러 시점의 평가 접근방법

라. 평가 결과 기록

발견된 문제와 결정 결과 또는 변화에 오해를 해소하기 위하여 평가 결과를 명확하고 정형화하여 기록하는 게 중요하다.

(1) 접근 방법

(가) 회의록

회의 진행을 기록한다. 명확하게 기술이 되면 명확한 토론 기록과 결과를 제공한다. 중요하지 않는 사항에 대하여 자세하게 기술하여 중요한 논점을 놓칠 수 있다.

(나) 결정 로그

평가 결과를 누가 작성하고 동의 했는지 와 논리적 근거를 결정 로그를 이용하여 기록한다. 간단하게 작성할 수 있으나 내용이 부족하고 주의 깊게 유지해야 한다.

(다) 검토 기록

합리적인 표준 형태로 문제나 해결책을 기록할 수 있다.

(라) 평가 기록

아키텍처 설계자와 이해당사자에게 유용하게 표준화되고 구조적인 방법으로 아키텍처 평가 실습 결과를 문서화한다.

(마) 문서 최종 점검

특정 결정이나 아키텍처 동의를 기록하는 일반적인 방법은 내용 일치를 기록한 문서를 최종 점검하는 것이다.

제 4 절 ATAM 프로세스

1. 개요

아키텍처 평가에는 크게 두 가지 측면을 고려한 방법이 있다. 첫째, 아키텍처를 위주로 평가하는 방법이 있다. 주요 프로젝트 의사 결정자가 수행하는 방법으로 아키텍처와 아키텍처를 정의하기 위해 내린 결정사항을 이해하는데 초점을 맞춘다. 둘째, 이해당사자를 위주로 평가하는 방법이 있다. 대표적인 이해당사자가 수행하는 방법으로 이해당사자를 참여시킴으로써 결과를 테스트한다 [2].

ATAM은 주로 다음과 같은 세 가지 경우에 사용된다. 첫째, 후보 아키텍처 선택을 평가하는데 사용된다. 둘째, 주요 업그레이드를 수행하기 이전에 기존 시스템을 평가하는데 사용된다. 셋째, 업그레이드를 할 것인지 교체를 할 것인지 선택하는데 사용된다.

2. 목적

ATAM은 아키텍처 결정사항의 결과를 다음과 같은 품질 속성 요구사항 측면에서 평가하기 위하여 사용한다.

o 위험을 발견하기 위하여 사용한다.

- 품질 속성에서 문제가 발생할 수 있는 결정사항을 찾는다.

o 위험이 발생하지 않는 것을 발견하기 위하여 사용한다.

- 비즈니스 목표를 실현화하는데 도움이 되는 품질을 높이는데 필요한 결정사항을 찾는다.

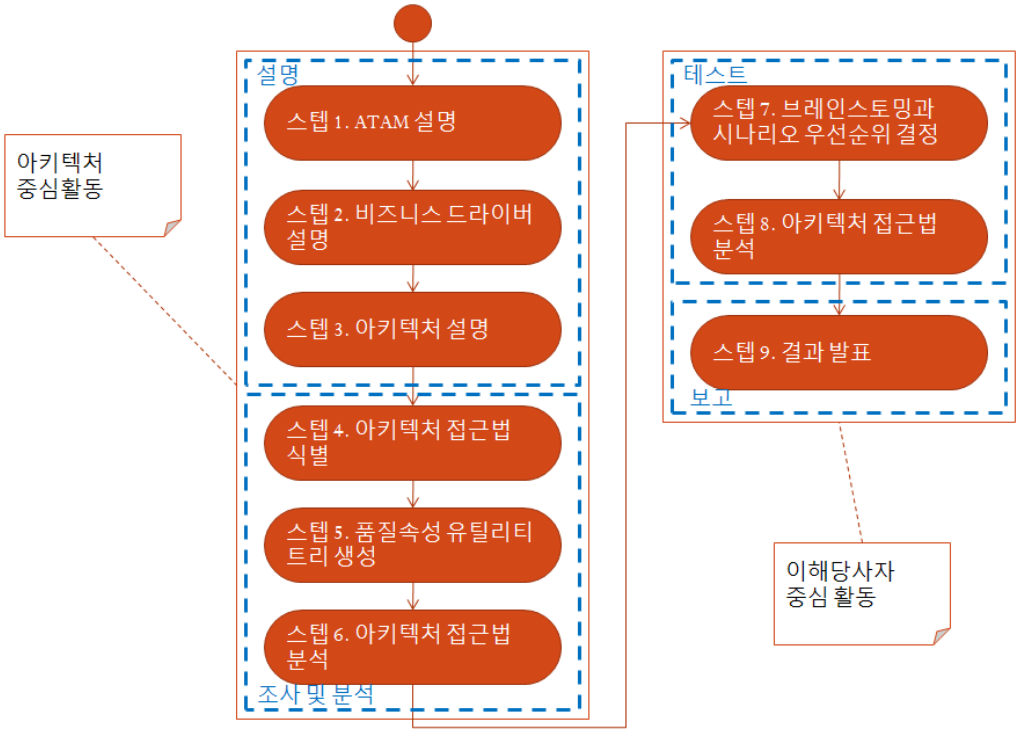
o 민감점을 발견하기 위하여 사용한다.

- 결정사항 중 일부 변화가 품질 특성에서 심각한 차이를 발생시키는 부분을 찾는다.

o 절충점를 발견하기 위하여 사용한다.

- 하나 이상의 품질 특성에 영향을 주는 결정사항을 찾는다.

3. ATAM 프로세스 구조



(그림 9) ATAM 프로세스 구조

ATAM 프로세스는 그림 9와 같이 4개의 그룹과 9개의 스텝으로 이루어져 있다. 첫 번째 그룹은 설명(presentation)으로 설명을 통하여 정보를 교환하기 위한 과정이다. 이 그룹에는 ATAM 설명, 비즈니스 드라이버 설명, 아키텍처 설명과 같은 3개의 스텝을 포함하고 있다. 두 번째 그룹은 조사(investigation) 및 분석으로 주요 품질 특성 요구사항과 아키텍처 접근법을 서로 비교 평가하기 위한 과정이다. 이 그룹에는 아키텍처 접근법 식별, 품질속성 유틸리티 트리 생성, 아키텍처 접근법 분석과 같은 3개의 스텝을 포함하고 있다. 세 번째 그룹은 테스트(testing)으로 관련 이해 당사자의 필요를 확정한 결과를 확인하기 위한 과정이다. 이 그룹은 브레인스토밍(brainstorm)과 시나리오 우선순위 결정, 아키텍처 접근법 분석

과 같은 2개의 스텝을 포함하고 있다. 네 번째 그룹은 보고(reporting)으로 ATAM 결과를 발표하기 위한 과정이다. 이 그룹에는 결과 발표 스텝을 포함하고 있다.

가. 스텝 1. ATAM 설명

이 스텝은 평가 리더가 프로젝트의 여러 대표들에게 ATAM을 소개하는 것이다. 모든 구성원이 지켜야 하는 프로세스에 대해 설명하고, 질의응답을 한다. 활동을 수행하기 위해 필요한 여건을 만들고 그 활동의 기대치를 정하는 것도 포함된다. 평가 리더는 ATAM의 단계와 평가 산출물을 간단히 설명한다. 산출물에는 시나리오, 아키텍처를 이해하고 평가하는데 사용된 질문, 유틸리티 트리, 식별된 아키텍처 접근법, 위험, 민감점 및 절충점이 포함된다.

나. 스텝 2. 비즈니스 드라이버 설명

이 스텝은 프로젝트 의사 결정자가 업무 관점에서 시스템 전반을 설명하는 것이다. 평가팀 구성원과 프로젝트 대표와 같은 평가 관련된 사람들은 시스템 배경과 개발 동기를 의미하는 핵심 비즈니스 드라이버를 이해해야 한다. 이를 위해서 다음과 같은 사항을 포함하여 설명한다.

- 시스템의 가장 중요한 기능
- 기술적, 관리적, 경제적, 정치적 제약사항
- 프로젝트와 관련된 비즈니스 목표와 문맥
- 주요 이해당사자
- 아키텍처 드라이버(주요 품질 특성 목표)

다. 스텝 3. 아키텍처 설명

이 스텝은 수석 아키텍처 설계자나 아키텍처 팀에서 수행하는 것으로 아키텍처를 적절한 수준으로 설명한다. 적절한 수준은 다음 요소에 의해 결정된다.

- 아키텍처 설계와 문서화 정도
- 쓸 수 있는 시간
- 행위 요구사항과 품질 요구사항의 특징

아키텍처 설계자는 사용하는 운영체제, 하드웨어, 미들웨어, 시스템과 상호 운영되는 시스

템과 같은 기술적 제약 사항에 대해 설명해야 한다. 특히, 아키텍처 설계자는 요구사항을 만족시키지 위한 아키텍처 접근법을 반드시 설명해야 한다.

라. 스텝 4. 아키텍처 접근법 식별

이 스텝은 아키텍처 설계자가 아키텍처 접근법과 스타일을 정리한다. 아키텍처 접근법은 시스템의 주요 구조를 정의한다. 또한, 시스템의 발전, 변화에 대한 반응, 공격에 대한 저항, 다른 시스템과의 통합과 같은 방법을 기술한다. 아키텍처 스타일은 컴포넌트 유형과 토폴로지(topology)을 기술하고, 컴포넌트 간의 데이터 패턴과 제어 상호작용을 기술한다. 또한 장단점에 대한 비공식적인 내용을 기술한다.

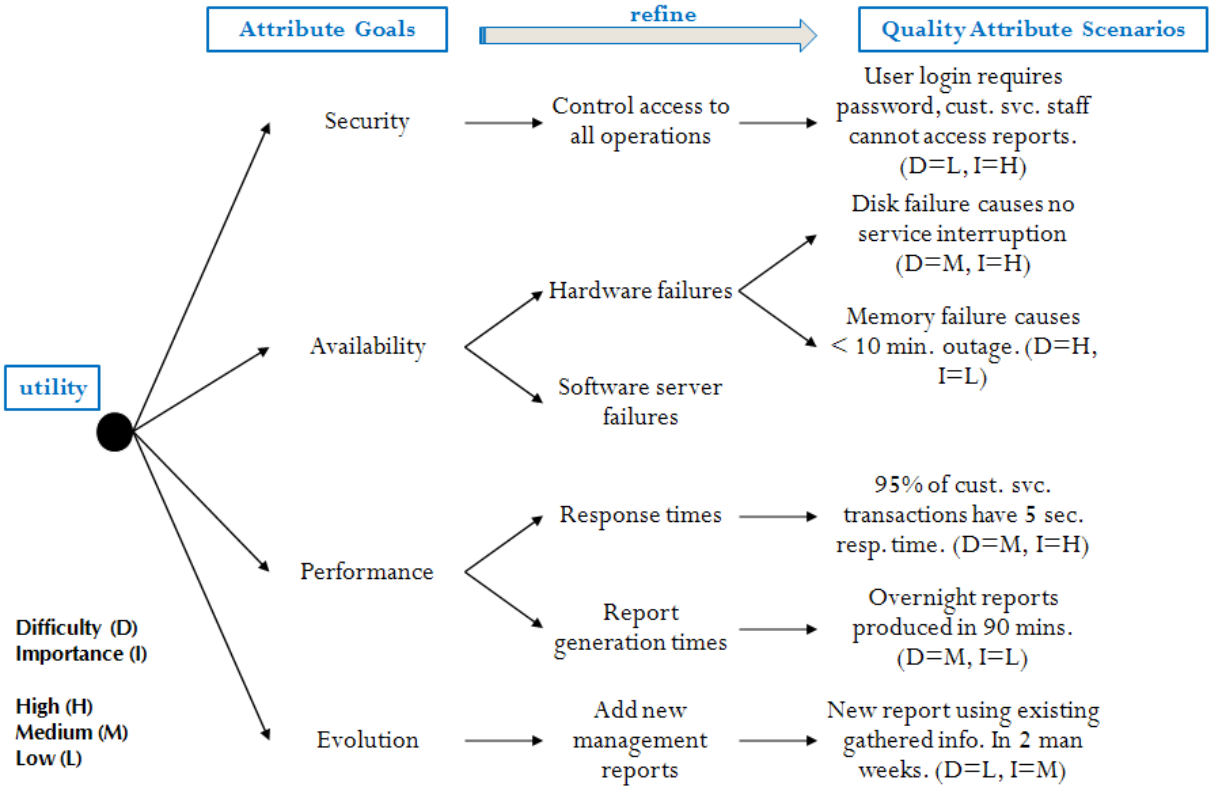
마. 스텝 5. 품질속성 유틸리티 트리 생성

(1) 수행 절차

- 시스템 활용도를 포함하는 품질 속성을 추출한다.
- 시나리오 수준에 맞게 품질 속성을 명시한다.
- 자극과 응답으로 품질 속성에 주석을 단다.
- 시나리오의 우선순위 점수를 매긴다.

(2) 유틸리티 트리

o 시스템 비즈니스 드라이버를 실제 품질 속성 시나리오로 직접적이고 효율적으로 바꾸는 메커니즘이다.
o 실제 품질 속성 요구사항을 생성하고 우선순위 점수를 하는데 도움을 준다.
  - 우선순위 기준
    · 성공적인 시스템을 만드는데 각 시나리오에서 중요한 역할을 한다.
    · 시나리오 구현에 따라 어려운 정도가 결정된다.
o 구조
  - 상위 수준 노드
    · 상위 수준 노드에는 성능, 변경성, 보안, 가용성과 같은 중요한 품질 목표가 존재한다.
  - 말단 노드
    · 말단 노드는 정제된 비즈니스 목표나 품질 목표가 포함된 시나리오가 존재한다.
o 유틸리티 트리 예제



(그림 10) 유틸리티 트리 예제

바. 스텝 6. 아키텍처 접근법 분석

(1) 수행 절차

- o 스텝 5에서 인식된 상위 우선순위 시나리오를 기반으로 시나리오에서 나타난 아키텍처 접근법을 추출하고 분석한다.
- o 아키텍처 위험, 무위험, 민감점, 절충점을 인식한다.

(2) 산출물

- o 아키텍처 접근법
  - 상위 우선순위 유틸리티 트리 시나리오와 관련된다.
  - 아키텍처 설계자는 자신이 선택한 접근법의 구성요소, 구성요소 사이의 관계와 제약사항까지 정리해 놓아야 한다.
- o 아키텍처 접근법에 대한 분석 질의
- o 분석 질의에 대한 아키텍처 설계자의 답변
- o 위험/무위험, 민감점, 절충점
  - 위험/무위험
    - 위험은 잠재적으로 불확실한 아키텍처 결정이다.

· 무위험은 내재적 가정을 기반으로 하는 정당한 결정이다.

· 위험/무위험은 아키텍처 결정, 품질 속성 요구사항, 논리적 근거로 구성되어 있다.

- 민감점

· 특정 품질 속성을 얻기 위한 아주 민감한 하나 이상의 컴포넌트가 가지는 속성이다.

· 민감점은 후보 위험이다.

· 동시에 여러 클라이언트가 데이터베이스에 접근할 경우 초당 처리하는 데이터베이스 수에 영향을 받는다. 이러한 문제는 성능에 있어서의 민감점이다.

· 데이터베이스 백업을 유지하는 것은 신뢰성에 영향을 준다.

- 절충점

· 하나 이상의 매개변수나 민감점에 영향을 주는 속성이다.

· 위험은 후보 절충점이다.

· 데이터베이스 백업을 유지하는 것은 성능에 영향을 미치기 때문에 신뢰성과 성능 사이에 절충이 필요하다.

[표 1] 아키텍처 접근법 분석 템플릿

Analysis of Architectural Approach				
Scenario #: No.	**Scenario** : text of scenario from utility tree			
Attribute(s)	Quality attribute(s) with which this scenario is concerned			
Environment	Relevant assumptions about the environment in which the system resides, and the relevant conditions when the scenario is carried out			
Stimulus	A precise statement of the quality attribute stimulus			
Response	A precise statement of the quality attribute responses			
Architectural Decision	**Sensitivity**	**Tradeoff**	**Risk**	**Nonrisk**
Architectural decisions relevant to this scenario that affect quality attribute response	Sensitivity Point #	Tradeoff Point #	Risk #	Nonrisk #
Reasoning	Qualitative and/or quantitative rationale			
Architectural Diagram	Diagram of diagrams of architectural views			

사. 스텝 7. 브레인스토밍(Brainstorm)과 시나리오 우선순위 결정

(1) 수행 절차

- o 이해당사자 그룹으로부터 시나리오 세트를 추출한다.
- o 모든 이해당사자가 참석하여 투표를 통하여 시나리오 세트의 우선순위를 결정한다.

(2) 시나리오 처리

- o 시나리오는 이해당사자의 관심을 표현하는 수단이다.
- o 시나리오는 품질 속성 요구사항을 이해하는데 도움이 된다.

(3) 좋은 시나리오

- o 시나리오에 영향을 주는 것이 무엇이며 이해당사자의 관심에 대한 해답이 무엇인지 확실히 하는 것이 좋은 시나리오이다.

(4) 브레인스토밍을 한 시나리오 종류

(가) 유스케이스 시나리오

- o 이해당사자는 시스템이 하는 일을 표현하는 방법이다.
- o 예를 들면, 원격 사용자는 최대 사용기간중에 웹을 통해서 데이터베이스 보고서를 요구하고 5초 안에 응답을 받기를 원한다.

(나) 성장 시나리오

- o 아키텍처는 충분히 예견할 수 있는 시스템의 변경을 표현한다.
- o 예를 들면, 새로운 데이터 서버를 추가하면 한 사람이 시나리오를 처리하기 위하여 대기하는 시간이 1~2.5초 정도 줄어든다.

(다) 사전탐사 시나리오

- o 절대 일어나지 않은 시스템의 변경을 표현한다.
- o 예를 들면, 모든 시스템의 가용성에 영향을 주지 않는 보통 운영 시에 반 이상의 서버가 고장이 난다.

아. 스텝 8. 아키텍처 접근법 분석 반복

- o 아키텍처 평가 리더는 스텝 7에서 찾아낸 최우선 시나리오를 아키텍처 설계자가 처리할 수 있도록 지침을 제공한다.
- o 스텝 7에서 새롭게 찾은 시나리오에 알맞은 아키텍처 접근법을 찾아서 스텝 6을 반복한다.

자. 스텝 9. 결과 발표

- o ATAM 평가 기간 동안에 수집된 정보를 기반으로 하여 ATAM 팀에게 수집한 정보를 발표한다.
- o 아키텍처 평가 리더는 ATAM 모든 과정을 이해당사자에게 상기시키고 산출물을 설명한다.
- o 산출물을 얻기 위한 방법
  - 문서화된 아키텍처 접근법
  - 시나리오와 우선순위
  - 매개변수 기반 질문
  - 유틸리티 트리
  - 발견된 위험
  - 문서화된 무위험
  - 민감점과 절충점 확인

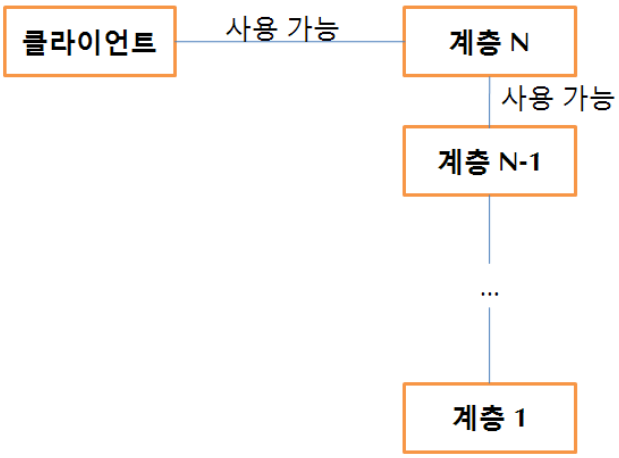
제 3 장 소프트웨어 아키텍처 스타일

본 장에서는 대표적인 아키텍처 스타일에 관해 설명한다. 각 아키텍처 스타일 별로 정의, 적용될 수 있는 상황, 구조, 장·단점 분석에 대해 기술한다.

제 1 절 Layered 스타일

1. 개요

계층 스타일(Layered Style)은 계층적으로 조직화될 수 있는 서비스로 구성된 어플리케이션에 적용하기 적합하다. 계층 스타일은 다음과 같은 4가지 항목(요소, 관계, 요소의 속성, 토폴로지)으로 기술될 수 있다. 계층 스타일의 요소는 계층이다. 각 계층은 모듈들의 응집된 집합이다. 계층내의 모듈들은 호출 또는 접근 될 수 있다. 계층 간의 관계는 사용 가능(Allowed to Use)의 관계이다 [3].



(그림 11) 클라이언트와 계층간의 관계

그림 11은 클라이언트와 계층간의 관계를 보여준다. 클라이언트는 가장 최상위 계층이 계층N에 제공하는 서비스를 사용할 수 있고, 계층은 N의 서비스를 클라이언트에 제공하기 위해서 하위 계층의 서비스를 사용한다.

두 계층에 대하여 이러한 관계가 설정되었다면, 첫 번째 계층에 있는 모듈은 두 번째 계층에 있는 모듈을 사용할 수 있다. 각 요소들은 4가지 속성(계층의 이름, 계층의 내용, 각 계층의 관계 기술, 계층의 응집도)을 기술한다. 각 계층에는 고유한 이름을 기술한다. 계층의 내용은 각 계층별로 포함하고 있는 모듈들을 기술한다. 각 모듈들은 한 계층에만 포함된다. 각 계층의 관계 기술은 각 계층이 사용할 수 있도록 허락된 계층은 무엇이며, 현재 계층은 어떤 계층에게 사용될 수 있는지의 관계를 기술한다. 계층의 응집도는 각 계층이 제공

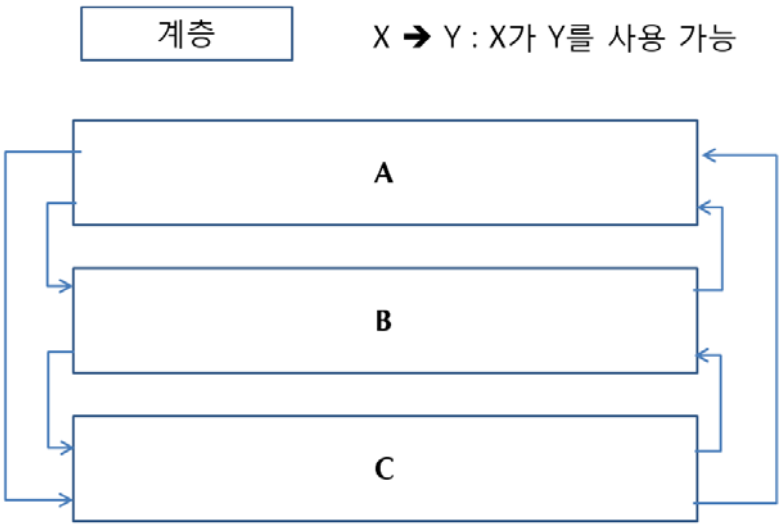
하는 응집된 기능성에 대하여 기술한다. 토폴로지는 각 요소들의 물리적인 관계를 기술한다.

2. 구조

계층(Layer)화된 스타일에서는 소프트웨어를 계층이라는 단위(Unit)로 나눈다. 각 계층은 하나의 가상 머신(Virtual Machine)으로도 표현된다. 여기서 가상 머신은 추상적인 컴퓨팅 장치(Abstract Computing Device)로 소프트웨어와 하드웨어 사이의 인터페이스(Interface) 역할을 하는 프로그램이다. 계층 스타일은 계층 다이어그램으로 표현된다. 각 가상 머신은 응집된 서비스(cohesive service)를 제공하는 공개 인터페이스를 포함한다.

계층 스타일에서 계층들 사이에는 계층 간의 상호작용하기 위한 순서 관계(ordering relation)가 존재한다. 상위 계층인 A, 하위 계층인 B, 두 개의 계층이 존재한다면, 계층 A는 계층 B에서 제공하는 공개 인터페이스의 서비스를 사용할 수 있다. 그러나, 일반적인 경우에는 하위 계층 B가 상위 계층 A에서 제공되는 공개 인터페이스의 서비스를 사용할 수 없다. 순서 관계를 정의함에 따라서, 하위 계층에서 상위 계층의 서비스를 사용할 수 있다.

그림 12는 계층 아키텍처의 예제이다. 그림 12에서 소프트웨어는 A, B, C 3개의 계층으로 구성되어 있고, 상위의 계층은 하위의 계층에서 제공되는 서비스를 사용할 수 있는 순서 관계뿐만 아니라, 하위 계층에서 상위 계층에서 제공되는 서비스를 사용할 수 있는 순서 관계이다.



(그림 12) 계층 아키텍처

많은 계층 시스템에서 하위 계층의 모듈이 상위 계층의 모듈을 사용하는 경우가 있는데, 이런 경우 상위 계층의 모듈을 사용할 수 있도록 순서 관계가 정립되어야 한다. 또한 최상위 계층의 모듈에서 최하위 계층의 모듈을 직접적으로 접근하여 사용할 경우에도 순서 관계가 정립되어야 한다. 계층 다이어그램에서 이러한 경우가 고려되어야 한다. 그러나 많은

경우에서, 최상위 계층에서 최하위 계층으로 직접 접근 또는 하위 계층에서 상위 계층으로 접근은 잘된 설계라 볼 수 없다.

계층들은 이식성(Portability)를 증진시키기 위해서 사용된다. 인터페이스는 특정 플랫폼에 종속적인 기능들(functions)을 노출시키지 않는다. 이러한 기능들은 플랫폼에 독립적인 추상 인터페이스에 의해 감춰진다. 하위계층에서는 작은 단위 기능 구현에 초점되어있다. 하위계층은 통신 채널, 분산 메커니즘, 프로세스 디스패치와 같은 기능들이 구현된다. 이와는 달리 상위 계층에서는 플랫폼 독립적인 기능이 구현된다.

3. 장·단점 분석

계층은 정보은닉(Information Hiding) 원칙이 적용된다. 하위계층의 변화는 상위계층의 인터페이스에 영향을 미치지 않는다. 계층 구조의 장점은 높은 이식성을 제공 하는 것이다. 머신, 운영체제, 또는 다른 하위 레벨 의존성은 레이어 안으로 숨겨진다. 따라서, 상위계층의 인터페이스가 변경되지 않는한, 상위 수준의 의존성은 인터페이스만 의존하게 된다. 계층 스타일의 장점을 정리하면, 효과적인 개념의 분리(Seperation of Concern)가 가능해 진다. 시스템을 잘 정의된 레벨로 추상화(Well-defined Level of Abstraction)를 시킬 수 있다. 변화가 발생했을 때 계층 인터페이스에 영향을 주지 않는다면, 변화에 대한 영향을 감소시킬 수 있다.

단점은 추가적인 실행 시 오버헤드가 발생할 수 있다. 또한 계층내에 사용되지 않는 서비스를 포함할 수 있다. 너무 많은 계층을 분할함으로써 성능이 감소에 영향을 줄 수 있다. 기능성을 계층에 할당할 때 올바르고 적절한 계층에 할당하기 힘들다.

제 2 절 Model-View-Controller (MVC) 스타일

1. 개요

모델 뷰 컨트롤러 (Model-View-Controller, MVC) 아키텍처 패턴은 상호작용 어플리케이션을 세 개의 컴포넌트로 구분한다. 모델 컴포넌트는 핵심 기능성과 데이터를 포함한다. 뷰는 사용자에게 정보를 보여준다. 컨트롤러는 사용자 입력을 다룬다. 뷰는 컨트롤러와 함께 사용자 인터페이스를 구성한다. 변화 전파 메커니즘은 사용자 인터페이스와 모델간의 일관성을 보장한다 [4].

예를 들어 선거 결과 분포를 보여주는 정보시스템에서 핵심데이터를 파이 차트, 막대차트, 스프레드 시트 등의 다양한 그래프 형태로 보여주는데 사용될 수 있다.

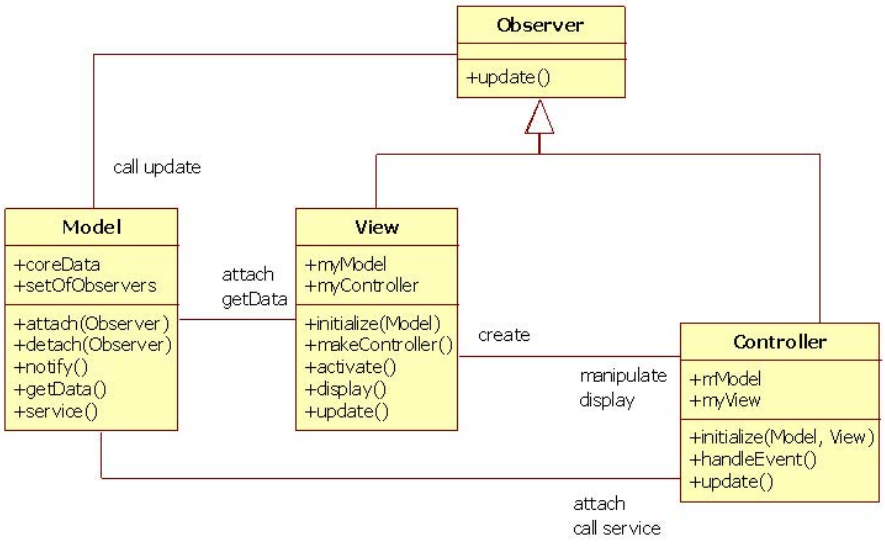
2. 구조

모델 컴포넌트는 어플리케이션의 핵심 기능을 포함한다. 모델 컴포넌트는 적절한 데이터를 캡슐화 하고 어플리케이션에 특화된 프로세싱을 수행한다. 컨트롤러는 사용자를 대신하여 이러한 프로시저들을 호출한다. 모델은 뷰 컴포넌트에 의해서 사용되게 될 데이터를 엑세스 하기위한 기능을 제공한다.

뷰 컴포넌트는 사용자에게 정보를 보여준다. 다양한 뷰는 다양한 방법으로 모델의 정보를 보여준다. 각각의 뷰는 변화 전파 메커니즘에 의하여 활성화될 갱신 프로시저를 정의한다. 갱신 프로시저가 호출되면 뷰는 모델로부터 보여지게 될 현재 데이터를 복구하고 그것들을 스크린에 출력한다.

컨트롤러 컴포넌트는 이벤트로서 사용자의 입력을 받아들인다. 어떻게 이러한 이벤트들을 컨트롤러에게 전달하는지는 사용자 인터페이스 플랫폼에 의존적이다. 이벤트들은 모델 혹은 연관된 뷰를 위한 요구사항들로 해석된다.

그림 13은 MVC 아키텍처의 구조를 보여준다. MVC 아키텍처는 크게 모델 클래스, 뷰 클래스, 컨트롤러 클래스를 가진다.



(그림 13) MVC 아키텍처의 구조

3. 장·단점 분석

o 장점

- 동일한 모델에 대한 다양한 뷰: MVC는 사용자 인터페이스 컴포넌트로부터 모델을 엄격하게 분리한다. 그러므로 다양한 뷰는 단일 모델과 함께 구현될 수 있다. 런타임

에 다양한 뷰는 동시에 열릴 수 있다. 그리고 뷰는 동적으로 열리고 닫힐 수 있다.

- 뷰 동기화: 모델의 변화 전파 매커니즘은 연관된 관찰자들이 적절한 시간에 어플리케이션의 데이터 변경을 통보 받을 수 있음을 보장한다. 이것은 모든 독립된 뷰와 컨트롤러를 동기화 시킨다.

o 단점

- 복잡도 증가: MVC 구조는 상호작용 어플리케이션을 구현하는데 항상 적합하지는 않다. 간단한 메뉴나 단순 텍스트를 위하여 모델과 뷰와 컨트롤러를 분리하는 것은 유연성을 얻지 못하며 복잡도를 증가시킨다.
- 잠재적인 많은 수의 갱신: 만약 단일 사용자의 행동이 많은 갱신을 초래한다면, 모델은 필요하지 않은 변화를 생략해야만 한다. 이것은 모든 뷰가 모델에 의한 모든 변화에 관심이 있지 않을 수 있기 때문이다. 예를 들어, 최소화된 윈도우즈 창은 해당 창이 일반 크기로 복구되기 전까지 갱신될 필요가 없다.

제 3 절 Client-Server 스타일

1. 개요

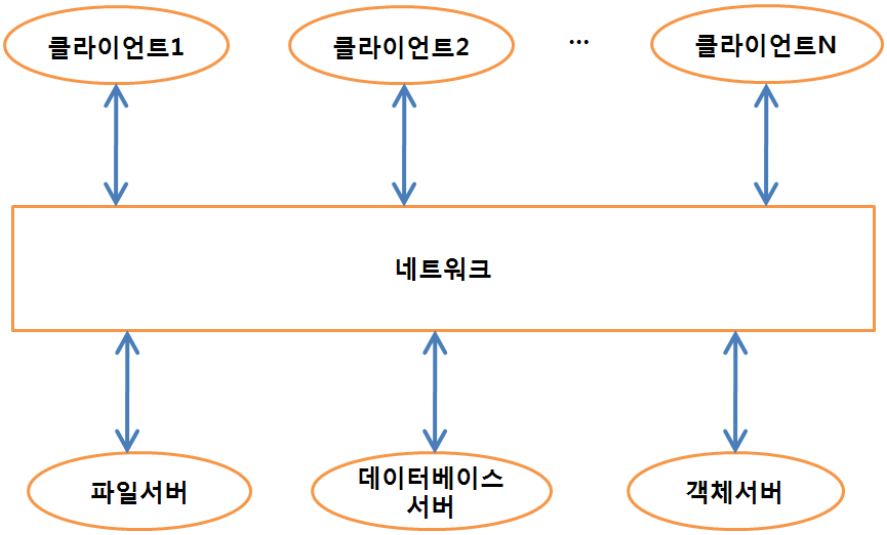
클라이언트-서버 스타일은 여러 컴포넌트는 걸쳐서 데이터와 데이터를 처리하는 프로세싱 부분이 분산되어 있는 어플리케이션에 적용하기 적합하다.

클라이언트-서버 스타일에서 컴포넌트 다른 컴포넌트의 서비스 요청에 의해서 상호작용한다. 이 스타일에서 중요한 점은 커뮤니케이션(Communication)은 클라이언트에 의해서 시작된다는 것이다. 클라이언트로부터 서비스 요청은 서비스를 제공하는 서버와 한 쌍을 이룬다. 이 스타일에서 서버는 하나 이상의 인터페이스를 통해 여러 서비스를 제공한다. 그리고 클라이언트는 시스템의 다른 서버가 제공하는 서비스들을 사용할 수 있다. 클라이언트-서버 스타일에서 하나의 중앙 서버 또는 분산된 여러 서버가 존재할 수 있다.

2. 구조

클라이언트-서버 스타일에서 컴포넌트 종류는 클라이언트와 서버로 구분할 수 있다. 그림 14는 클라이언트의 개략적인 구조를 나타낸다. 클라이언트는 서버에서 제공하는 서비스를 호출하는 컴포넌트이고, 서버는 프린트, 데이터 관리등 과 같은 구체적인 서비스를 제공하는 독립 컴포넌트이다. 커넥터는 클라이언트가 원격지에 있는 서버를 접속하기 위해 사용되는 네트워크인데, 클라이언트-서버 스타일을 위한 커넥터 종류(Connector Type)는 서비스를 호출하기 위한 요청-응답 커넥터(request-reply connector)가 사용된다. 서버는 서버가 제공

하는 여러 서비스를 포함하는 인터페이스를 정의한다. 서비스 호출 계층구조(Hierarchy)에서, 서버는 다른 서버로부터 서비스를 요청하는 클라이언트로써 역할을 할 수 있다. 클라이언트-서버 스타일에서 사용될 수 있는 커넥터 타입은 요청-응답 커넥터와 비대칭적 호출(Asymmetric Invocation)이다. 일반적인 클라이언트-서버 시스템의 동작 방식은 비대칭적이다. 클라이언트는 서버의 서비스를 요청함으로써 서비스를 실행시킨다. 그러므로 클라이언트는 호출할 서비스를 알고 있어야 하고, 클라이언트는 모든 상호작용을 시작해야 한다. 이와 달리, 서버는 서비스의 요청 전에 미리 클라이언트를 알 수 없고, 클라이언트 요청에 응답해야 한다.



(그림 14) 클라이언트-서버 스타일의 구조

서비스 호출의 다른 형태는 동기 호출이다. 요청한 서비스의 결과 값이 제공되어 완료될 때까지 서비스의 요청자는 기다리거나 블락된다. 클라이언트-서버 스타일의 사용에 대한 제약사항은 서버에 접속할 수 있는 클라이언트 수의 제한이 될 수 있거나 또는 서버가 다른 서버와의 연동에 제약이 있을 수 있다. 클라이언트-서버 스타일의 특화된 사례는 n-티어 클라이언트-서버 모델(n-tiered Client-Server Model)이다. 이 경우, 클라이언트와 서버는 n 레벨 계층구조를 이루고 상위 티어는 하위 티어의 서비스를 호출하는 클라이언트로 구성된다. N-티어 시스템은 일반적으로 비즈니스 처리 어플리케이션에서 적용되며, 주로 3개 티어로 구성된다. 첫째 티어는 클라이언트 어플리케이션으로 구성되고, 둘째 티어는 비즈니스 로직 서비스, 셋째 티어는 데이터 영구성, 동시접근 관리, 쿼리 지원 등의 데이터 관리 서비스로 구성된다.

3. 장·단점 분석

클라이언트-서버 스타일의 장점은 직접적으로 데이터를 분산시키고, 위치 투명성

(Transparency of Location)을 제공한다. 클라이언트 어플리케이션을 서버가 제공하는 서비스로부터 분리시킨다. 이 스타일은 공통적인 서비스를 분리시킴으로써 시스템의 이해를 돕는다. 기능성들을 그룹화함으로써 시스템을 하드웨어 플랫폼에 배치(Deploy)하는데 유용하게 사용된다. 기능성을 클라이언트와 서버로 분리함으로써 기능성들을 독립적인 티어에 할당할 수 있다. 그러므로, 성능적 확정성과 신뢰성을 얻을 수 있다. 또한 기존의 새로운 서버를 추가하거나 업그레이드하기 용이하다.

클라이언트-서버의 단점은 서비스와 서버의 이름을 관리하는 중앙 레지스터가 없기 때문에 어떤 서비스가 이용가능한지 찾기가 힘들다.

제 4 절 Pipes and filters 스타일

1. 개요

파이프 필터 아키텍처 스타일 (Pipes and Filters Style) 은 데이터의 스트림을 처리하는 시스템을 위한 구조를 제공한다. 각 프로세싱 시스템은 필터 컴포넌트 내에 캡슐화 되어있다. 데이터는 인접 필터사이의 파이프를 통하여서 전달된다. 필터의 재조합은 관련된 시스템의 집단을 생성하게 해준다.

예를 들어 유닉스의 파이프는 유닉스 프로세스들을 연결하게 해주고, 다양한 필터를 이용한 이미지 처리 프로그램, 시그널 프로세싱, 음향 비디오 스트리밍에 사용될 수 있다.

2. 구조

필터컴포넌트는 파이프라인의 프로세싱 단위다. 필터는 그것의 입력 데이터를 풍부하게고 정제하고, 변화시킨다. 이것은 계산과 정보 추가를 통해서 데이터를 풍부하게하고 정보를 추출하거나 응축함으로서 데이터를 정제하고, 데이터를 다른 형태로 표현함으로서 변화시킨다. 구체적 필터는 다음과 같은 세 가지 기본 원칙에 따라 조합함으로서 구현될 수 있다.

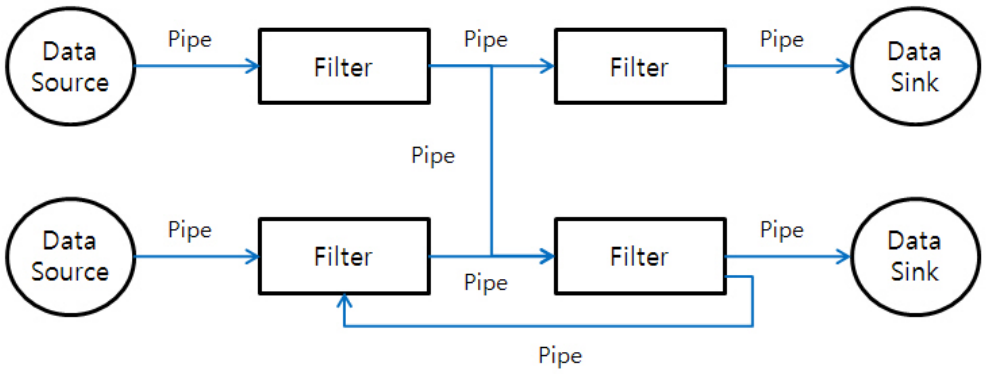
- 뒤에 이어지는 파이프라인 요소는 필터로부터 출력데이터를 가져온다.
- 이전의 파이프라인 요소는 필터에게 새로운 입력 데이터를 전달한다.
- 일반적으로 필터는 루프(loop)형태로 활성화 된다.

파이프는 데이터 소스와 첫 번째 필터 간, 마지막 필터와 데이터 싱크 간의 필터들 간 연결을 표현한다. 만약 활성화된 두 개의 컴포넌트들이 참여한다면 파이프는 활성화된 두 개의 컴포넌트를 동기화 시킨다. 해당 동기화는 FIFO(First-in-first-out) 필터를 이용하여 수행된다.

데이터 소스는 시스템으로의 입력을 표현하고 동일 구조나 동일 타입의 데이터 값의 순

서를 제공한다.

데이터 싱크는 파이프라인의 마지막으로부터 결과를 수집한다. 두 가지 다른 데이터 싱크가 가능하다. 엑티브 데이터 싱크는 이전 프로세싱 단계로부터 결과를 가져오고 반면에 패시브 데이터 싱크는 이전 필터에 결과를 입력하는데 사용된다.



(그림 15) 파이프 필터 아키텍처 구조

3. 장·단점 분석

o 장점

- 필터 교환을 통한 유연성: 필터는 파이프라인 프로세싱 내에서 손쉬운 교환을 가능하게 하는 단순한 인터페이스를 가진다. 필터 컴포넌트가 서로를 직접적으로 호출하는 경우에도 교환 필터 컴포넌트는 쉬운 처리가 가능하다.
- 재조합을 통한 유연성: 필터 컴포넌트들의 재사용성을 이용한 재조합은 필터를 재배열 하거나 혹은 새로운 필터를 추가함으로써 새로운 프로세싱 파이프라인을 만들 수 있게 해준다.

o 단점

- 상태정보를 공유하는 것은 비용이 많이 들거나 유연하지 못함: 만약 프로세싱 스테이지가 많은 양의 전역 데이터를 공유할 필요가 있다면 파이프 필터 패턴을 적용하는 것은 효과적이지 못하거나 패턴을 이용함으로서 얻는 이득이 존재하지 않을 수 있다.
- 데이터 변화 과부하: 높은 유연성 결과를 달성하기 위한 모든 필터의 입력과 출력을 위한 단일 데이터 타입의 사용은 데이터 변환 과부하를 일으킨다.

제 5 절 Publish-Subscribe 스타일

1. 개요

퍼블리쉬-서브스크라이브(Publish-Subscribe) 스타일에서, 컴포넌트는 공표된(Announced) 이벤트를 통해서 동작한다. 컴포넌트는 여러 이벤트를 구독(Subscribe)할 수 있다. 퍼블리쉬-서브스크라이브 런타임 인프라스트럭처(Publish-Subscribe Runtime Infrastructure)의 역할은 각 공표된 이벤트가 모든 서브스크라이버에게 전달하는 것이다. 따라서 이 스타일에서 주된 커넥터 형태는 이벤트 버스(Event Bus) 종류이다. 퍼블리쉬 컴포넌트는 이벤트를 공표함으로써 버스에 위치시키고, 커넥터는 이러한 이벤트를 적절한 서브스크라이브 컴포넌트에게 전달한다. 퍼블리쉬-서브스크라이브 스타일은 메시지 소비자와 생산자를 분리시키기 위해서 일반적으로 사용된다.

2. 구조

퍼블리쉬-서브스크라이브 스타일은 독립적인 프로세스 또는 객체로 구성된 시스템에 적합하다. 이 스타일에서 컴포넌트 개발자는 컴포넌트의 공표된 이벤트를 받을 서브스크라이버의 수를 미리 알 수 없기 때문이다. 퍼블리쉬-서브스크라이브 시스템의 기본 속성은 컴포넌트들을 분리시키고 시스템의 다른 부분에 영향 없이 시스템의 일부분을 수정할 수 있는 능력을 향상시킨다.

퍼블리쉬-서브스크라이브 스타일은 여러 가지 형태가 있을 수 있다. 가장 일반적인 형태는 묵시적 호출(Implicit Invocation)이라고 불리는데, 컴포넌트는 절차적인 인터페이스(Procedural Interface)를 가지고 있으며, 구독할 이벤트 타입을 등록한다. 이벤트가 공표될 때, 서브스크라이브 컴포넌트들의 프로시져(Procedure)가 런타임 인프라스트럭처에 의해서 결정된 순서에 의해서 호출된다.

비쥬얼 베이직과 같은 사용자 인터페이스 프레임워크는 묵시적 호출방식을 사용한다. 사용자 코드가 마우스 클릭과 같은 미리 정의된 이벤트와 사용자 코드가 연결됨으로써 프레임워크에 추가된다.

다른 퍼블리쉬-서브스크라이브 형태에서, 이벤트는 단순히 적절한 컴포넌트에 라우트된다. 이벤트를 어떻게 처리할 것인지는 컴포넌트의 작업이다. 이러한 시스템은 이벤트 스트림을 관리하기 위해서 개인적인 컴포넌트에 더 많은 책임을 위임한다.

3. 장·단점 분석

퍼블리쉬-서브스크라이브 스타일은 이벤트와 메시지를 알려지지 않은 수신자(Recipient)에게 보내기 위해 사용된다. 수신자가 알려지지 않았기 때문에, 새로운 수신자가 메시지 와 이벤트 생산자(Producer)의 수정 없이 추가 될 수 있다.

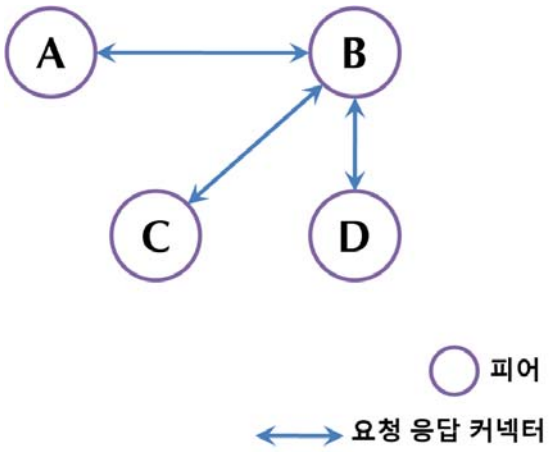
제 6 절 Peer-to-Peer 스타일

1. 개요

피어-투-피어(Peer-to-Peer) 스타일에서의 컴포넌트들은 서비스들을 교환하면서 피어로써 직접으로 상호작용을 한다. 피어-투-피어 커뮤니케이션은 클라이언트-서버 스타일에서의 요청/응답 상호작용의 한 종류이다. 어떠한 컴포넌트라도 서비스를 요청함으로써 다른 컴포넌트와 상호작용할 수 있다. 예를 들어, 피어-투-피어 시스템은 CORBA, COM+, Java RMI 등과 같은 분산된 객체 인프라스트럭처 기반의 아키텍처를 포함한다.

2. 구조

피어-투-피어 스타일에서 컴포넌트 타입은 객체, 분산 객체, 클라이언트등과 같은 피어들이다. 커넥터 타입은 절차적 호출(invoke-procedure) 커넥터이다. 클라이언트 서버와는 다르게, 상호작용은 양쪽 어느 피어에 의해서도 시작될 수 있다. 피어는 다른 피어로부터 요청을 하는 서비스와 현재 피어에서 제공하는 서비스를 기술하는 인터페이스를 가지고 있다. 피어-투-피어 시스템의 실행 흐름은 대칭적이다. 피어들은 다른 피어들로부터 서비스를 요청함으로써 다른 피어들과 협력하여 피어들이 성취하고자하는 작업을 시작한다. 피어-투-피어 스타일 사용의 제약 사항은 주어진 피어에 연결될 수 있는 피어들의 수가 제한되어 있다는 것이다.



(그림 16) 피어-투-피어 스타일

그림 16은 시스템이 피어-투-피어 스타일로 표현한 예이다. 그림에서 원은 시스템을 구성하는 피어를 나타내고, 화살표는 피어간의 요청 응답 커넥터를 의미한다. B는 A의 피어인 동시에 C와 D의 피어로써 각 피어와의 상호작용을 하고 있다.

3. 장·단점 분석

피어-투-피어 스타일은 협력의 영역(Area of Collaboration)으로써 시스템을 분할하는 뷰를 제공한다. 피어들은 다른 피어들과 직접적으로 상호작용을 하고, 클라이언트와 서버 모두의 역할을 할 수 있다. 이러한 분할은 시스템을 분산 시스템 플랫폼에 배치하기 위한 유연성을 제공한다. 피어들은 가장 최근 데이터를 접근하기 때문에, 서버로서 역할을 하는 컴포넌트의 부하를 줄여줄 수 있다. 또한 여러 피어로 분산되어 있기 때문에, 서버로서의 역할의 하기 위해서 더 많이 필요한 용량과 인프라스트럭처를 분산시킬 수 있다. 이러한 점은 데이터 업데이트를 위한 커뮤니케이션의 필요와 중앙 서버 스토리지를 위한 필요를 감소시킨다. 피어-투-피어 컴퓨팅은 분산 컴퓨팅 어플리케이션에서 사용된다. 적절한 배치는 어플리케이션이 CPU와 디스크 자원의 효율적인 사용을 할 수 있게 한다.

제 7 절 Blackboard 스타일

1. 개요

블랙보드 아키텍처 패턴은 결정적 해결 전략이 존재하지 않는 문제를 해결하는데 유용하다. 블랙보드 내에서 특화된 서브시스템은 서브시스템의 지식을 가능한 부분적인 혹은 개략적인 해결책을 세우기 위해 조합한다.

2. 구조

블랙보드 스타일은 블랙보드, 지식 자원의 집합, 컨트롤 컴포넌트로 구성된다.

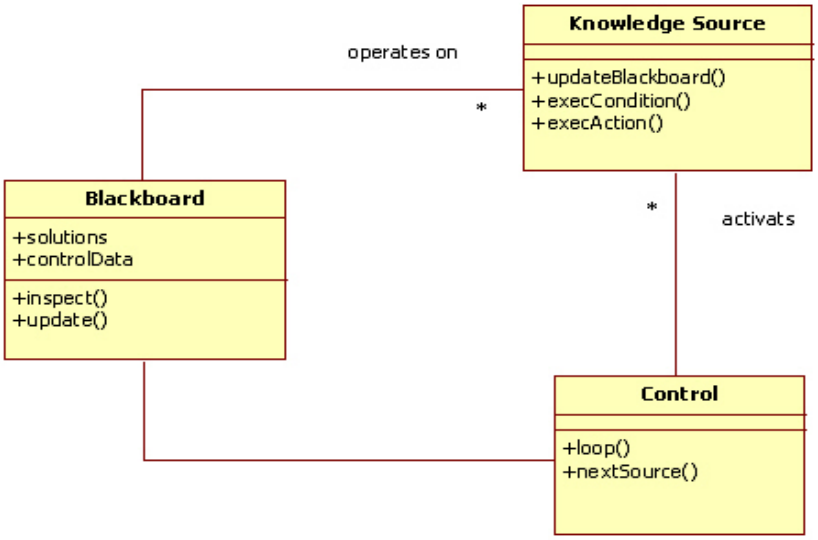
블랙보드는 중심 데이터 저장소 이다. 해결 공간 요소와 컨트롤 데이터가 이곳에 저장된다. 블랙보드에 나타낼 수 있는 데이터 요소들의 집합을 어휘 (Vocabulary)라는 용어를 사용하여 표현한다. 블랙보드는 블랙보드에 모든 지식 자원을 쓰고 읽을 수 있는 인터페이스를 제공한다.

지식 자원의 집합은 나누어져 있고 독립적인 서브시스템들은 전체적 문제의 특화된 측면을 해결하기위해 존재한다. 지식 자원의 집합은 함께 전체적 문제 영역을 모델링한다. 지식 자원 집합은 독립적으로 하나의 테스크를 해결 할 수 없다. 해결책은 다수의 지식 근원의 통합을 통하여 수립될 수 있다.

컨트롤 컴포넌트는 블랙보드의 변화를 감시하고 다음에 취해야할 행동을 결정하는 루프(loop)를 실행한다. 이것은 지식 근원을 평가하는 계획을 세우고 지식 어플리케이션 전략에

따라 지식 근원을 활성화 한다.

그림 17은 블랙보드 스타일의 구조를 보여준다. 블랙보드 스타일은 블랙보드, 지식 자원의 집합, 컨트롤 컴포넌트로 구성된다.



(그림 17) 블랙보드 아키텍처의 구조

3. 장·단점 분석

o 장점

- 다양한 접근법 제공: 적합한 접근법이 존재하지 않는 영역이나 해결책에 대한 완벽한 연구가 존재하지 않는 경우 블랙보드 패턴은 다양한 알고리즘을 통한 실험이 가능하다.
- 유지보수성, 가변성 제공: 각각의 지식 자원들과 알고리즘 그리고 중앙 데이터는 엄격하게 나누어져 있기 때문에 블랙보드 아키텍처는 유지보수성과 가변성을 제공한다. 그러나 모든 모듈은 블랙보드를 통하여서 통신이 가능하다.
- 재사용 가능한 지식 자원: 지식 자원은 특정한 테스크에 독립적이다. 블랙보드 아키텍처는 그것들을 재사용하는데 도움을 준다. 재사용의 전재조건은 해당 지식 기반과 기반이 되는 블랙보드 시스템이 동일한 프로토콜과 데이터를 이해 할 수 있어야 한다는 것이다.

o 단점

- 테스팅의 어려움: 블랙보드 시스템의 연산이 결정론적인 알고리즘을 따르지 않기 때문에 블랙보드 시스템의 결과는 재생산이 불가능하다.
- 완전한 해결책을 보장 못함: 일반적으로 블랙보드 시스템은 주어진 테스크의 일정 부분에 대한 해결책을 제시한다.

- 많은 개발 노력의 필요: 대부분의 블랙보드 시스템은 발전하는데 수년이 걸린다.
- 적합한 통제 전략 수립의 어려움: 컨트롤 전략은 직접적으로 설계될 수 없고 실험적 접근을 요구한다.

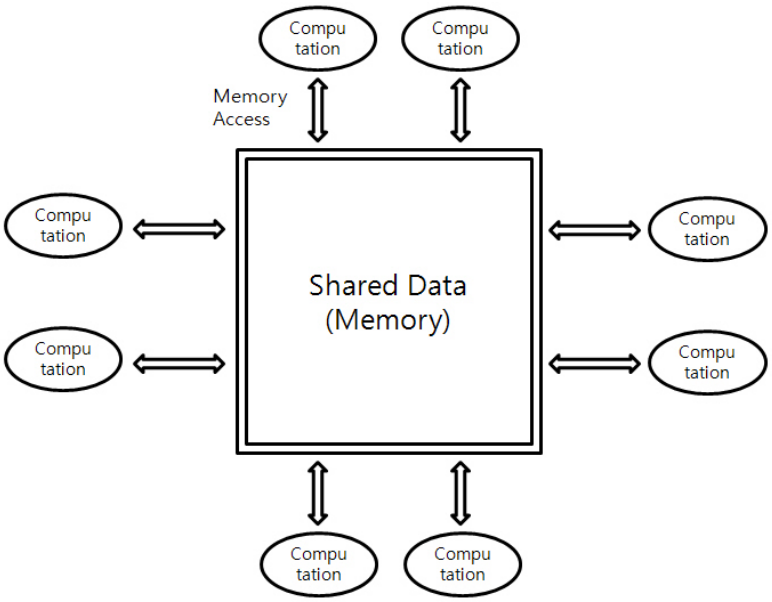
제 8 절 Repository 스타일

1. 개요

레파지토리 (Repository) 스타일은 중심이 되는 문제가 수립되고 논증되고 유지되는 복잡한 정보를 갖고 있는 어플리케이션에 적합한 스타일이다. 정보시스템, 프로그래밍환경, 그래픽 에디터, 인공지능 기반, 역공학 시스템 등에 이용 될 수 있다.

2. 구조

중앙 데이터 구조 컴포넌트는 시스템의 상태를 나타낸다. 독립적 컴포넌트의 집합은 중앙 데이터 구조에서 동작 한다. 그림 18은 레파지토리 스타일의 구조를 보여준다.



(그림 18) 레파지토리 스타일의 구조

3. 장·단점 분석

o 장점

- 대량의 데이터를 저장: 레파지토리 스타일은 대량의 데이터를 저장하는데 효과적이다.
- 공유모델 발행 가능: 공유모델은 레파지토리 스키마 형태로 발행이 가능하다.
- 중앙 집중화 관리: 중앙 집중화를 통한 데이터 백업이 용이하고 보안적인 측면에서 강하며, 동시성 조절이 가능하다.

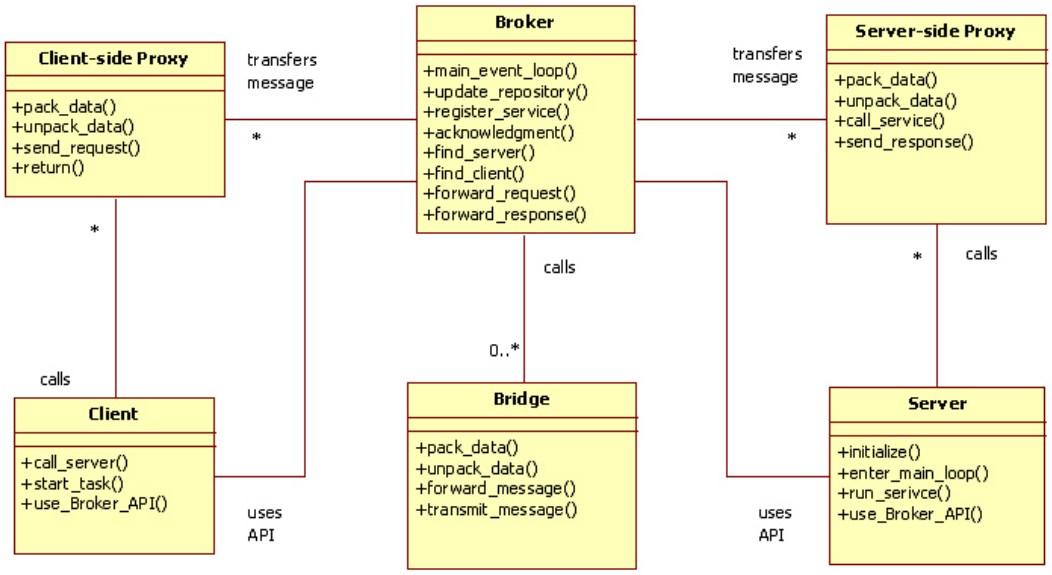
o 단점

- 데이터 모델의 사전 동의 필요: 사전에 레파지토리 스타일내에서 데이터 모델에 대한 동의가 필요하다
- 데이터 분산의 어려움: 데이터가 중앙 집중화 되어 관리되기 때문에 데이터를 분산하는데 어려움이 존재한다.
- 데이터 발전에 어려움이 있다.

제 9 절 기타

1. Broker 스타일

브로커 (Broker) 스타일 패턴은 원격 서비스 호출로 상호작용하는 분리된 컴포넌트로 구성되어 이는 분산 소프트웨어 시스템에 사용될 수 있다. 브로커 컴포넌트는 요청을 전송하거나, 결과 혹은 기댓값을 전송하는 등의 통신을 조화롭게 수행할 책임을 갖고 있다. 브로커 스타일의 일반적인 구조는 아래와 같다.



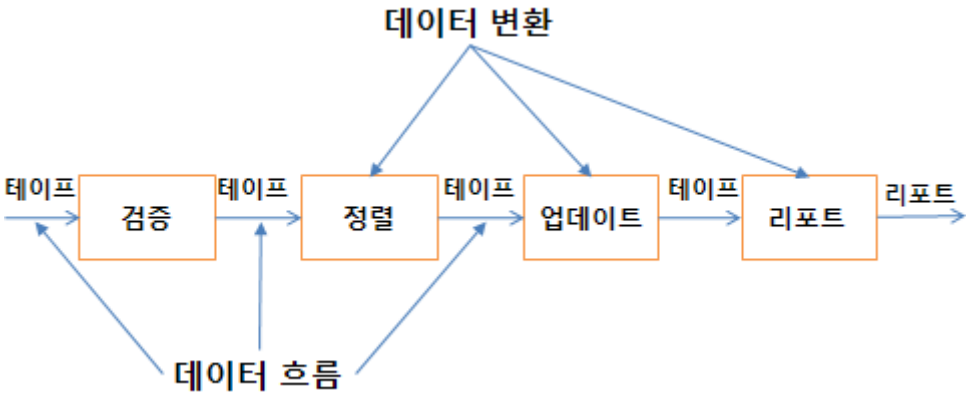
(그림 19) 브로커 스타일의 구조

브로커 스타일은 위치 투명성을 갖는다. 브로커가 클라이언트나 서버의 위치를 식별하기 때문에 클라이언트나 서버는 서로의 위치를 알 필요가 없다. 또한, 컴포넌트의 확장성을 갖는다.

브로커를 사용하기 때문에 제안된 효율성을 갖고 낮은 결함 허용성을 갖는다.

2. Batch sequential 스타일

배치 시퀀셜(Batch Sequential) 스타일은 파이프 엔 필터 스타일의 특화된 버전으로, 각 필터는 결과를 산출되기 전에 필터의 모든 입력 데이터를 처리한다. 그림 20은 배치 시퀀셜 스타일의 구조에 대한 예이다. 데이터 변환 스텝은 독립적인 프로그램이다. 다음 스텝이 시작 되기전에 각 스텝은 완료되어야 한다. 스텝들 사이에서 전송되는 데이터들은 큰 단위의 데이터들이 전송된다. 배치 시퀀셜은 전통적인 데이터 처리 어플리케이션에 적용된다.



(그림 20) 배치 시퀀셜 스타일의 구조

제 4 장 보안구조 (ADV_ARC)의 소프트웨어 공학적 분석

본 장에서는 CC에서 언급하고 있는 보안구조(ADV_ARC)가 만족해야 하는 속성에 관해 소프트웨어 공학적인 관점에서 분석한다. 소프트웨어 아키텍처가 공통적으로 만족해야 하는 속성인 아키텍처 타당성(Architectural Soundness), 보안 아키텍처에서 특별히 중요시되는 3가지 속성인 자체보호(Self-Protection), 영역분리(Domain Isolation), 우회불가성(Non-Bypassability)에 대해 분석한다. 마지막으로 보안 아키텍처 명세서 작성 및 평가 지침을 도출하기 위한 근거를 분석한다.

제 1 절 Architectural Soundness에 대한 해석

보안구조 (ADV_ARC) 패밀리는 개발자가 제공해야 하는 TSF의 보안 아키텍처 명세서에 관해 설명하고 있다 [5]. 보안 아키텍처 명세서를 통해 TSF가 만족해야 하는 속성이 만족되었는지 확인할 수 있으며, 나아가 TOE의 보안성 분석의 근거가 된다.

보안 아키텍처 명세서는 아키텍처 명세서 가운데 하나이다. 소프트웨어 공학 관점에서 아키텍처가 만족해야 하는 중요한 속성 가운데 하나가 아키텍처 타당성이다. 다음의 그림 21은 CC에서 아키텍처 타당성에 관해 인용한 문구이다.

219 The description of architectural soundness can be thought of as a developer's vulnerability analysis, in that it provides the justification for why the TSF is sound and enforces all of its SFRs. Where the soundness is achieved through specific security mechanisms, these will be tested as part of the Depth (ATE_DPT) requirements; where the soundness is achieved solely through the architecture, the behaviour will be tested as part of the AVA: Vulnerability assessment requirements.

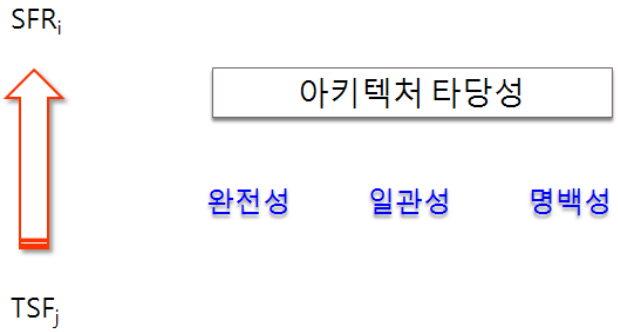
(그림 21) Architectural Soundness가 인용된 CC 원문

CC 본문에서는 아키텍처가 타당(sound)하게 명세되기 위해 가져야 하는 특성에 관해 언급하고 있다. 첫째는 TSF가 왜 타당한지에 대한 근거를 제시해야 한다는 것이고, 둘째는 TSF가 모든 SFR을 만족하고 있다는 것에 대한 정당성이 마련되어 한다는 것이다.

소프트웨어 공학 관점에서 아키텍처 타당성의 정의는 수학적 이론에 그 배경을 두고 있다. 아키텍처 타당성은 유효성(Validity)과 의미가 유사하며 세부 속성으로는 완전성(Completeness), 일관성(Consistency), 명백성(Unambiguity)이 있다.

그림 22와 같이 SFRi를 만족하는 TSFj를 설계할 경우, TSFj가 타당하는 것의 의미는 다음

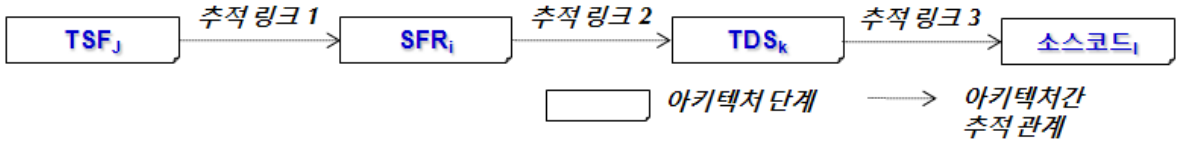
3 가지 조건을 만족함을 의미한다. 첫째, TSFj가 SFRi의 범위를 완전하게 지원해야 한다. SFRi의 요구사항 중에 TSFj에 반영되지 않은 누락된 정보가 있어서는 안된다. 둘째, SFRi의 내용이 TSFj에 일관성 있게 반영되어야 한다. SFRi에 있는 내용이 모두 TSFj에 반영되었다 하더라도 SFRi의 요구사항을 준수하지 않을 경우에는 TSFj가 타당하지 않은 것이다. 셋째, TSFj가 애매모호하지 않아야 한다. TSFj가 다른 의미로 해석되지 않도록 명백해야 한다.



(그림 22) 아키텍처 타당성의 세부요소

실제 적용할 경우에는 위의 3 가지 요소를 만족하도록 보안 아키텍처를 타당하게 명세하는 것은 쉽지 않다. 또한, 아키텍처 명세를 보고, 타당성을 평가하는 것에도 어려움이 있다.

일관성은 연관되고 조직된 아키텍처가 얼마나 잘 응집되어 있는지 평가하기 위한 요소이다. 각 단계의 아키텍처의 연결 관계가 명확해야 일관성이 좋아진다. 그림 23과 같이 TSF_j의 범위, 기술된 SFR_i, 설계된 TDS_k, 구현된 소스코드가 추적되어서 연결 관계가 명확하게 기술될 수 있어야 한다.



(그림 23) 일관성 맵

제 2 절 Self-Protection에 대한 해석

자체보호는 보안 아키텍처가 만족해야 하는 3가지 주요 속성 가운데 하나로써 그림 24에서 명시한 바와 같이 허용되지 않은 신뢰성이 없는 개체의 조작으로 인해 TSF가 변경되는 것을 자체적으로 보호하는 TSF의 능력을 의미한다. 여기서 중요한 점은 자체적으로 보호해야 한다는 특성으로, 자체보호는 자율 컴퓨팅(Autonomic Computing)에서 추구하는 주요 4가지 자율 관련 속성의 일환이다 [6]. 시스템이 악의적인 공격이나 단계적으로 발생하는 실패에 자동적으로 방어하는 속성을 의미한다. 시스템이 실패하기 이전에 실패가 일어날 것을

예상하고 조기에 실패를 예방한다.

513 *Self-protection* refers to the ability of the TSF to protect itself from manipulation from external entities that may result in changes to the TSF. Without these properties, the TSF might be disabled from performing its security services.

514 It is oftentimes the case that a TOE uses services or resources supplied by other IT entities in order to perform its functions (e.g. an application that relies upon its underlying operating system). In these cases, the TSF does not protect itself entirely on its own, because it depends on the other IT entities to protect the services it uses.

(그림 24) Self-protection이 인용된 CC 원문

자체보호를 적용한 아키텍처를 설계하기 위해서는 여러 가지 원리가 적용될 수 있다. 대표적으로는 관심의 분리(Separation of Concern, SOC) 원리가 적용될 수 있다. 객체지향(Object-Oriented) 개발 방법론에서는 인터페이스와 구현의 분리를 통해 SOC를 실현함으로써 관심간의 의존성을 낮추어 자체보호를 위한 범위를 특정 관심 영역으로 제한할 수 있다. 또한, 정보 은닉 원리를 적용하여 공용 인터페이스(Public Interface)를 두고 내부의 변수(Variable)나 리소스 접근하는 소스에 대한 접근을 제한함으로써 자체보호를 위한 기반을 마련할 수 있다. SOC와 비슷한 맥락으로 고도의 모듈화를 통해 모듈간의 coupling을 낮춰주는 것도 자체보호를 적용하기 위한 기본적인 설계라 할 수 있다.

자율 컴퓨팅에서는 자체보호를 위해 다음과 같은 4가지 작업을 수행한다 [7]. 첫째 작업은 모니터링(Monitoring)이다. 보호 대상 시스템을 자체적으로 모니터링 하는 센서 컴포넌트를 통해 보호 대상 시스템에 어떠한 변화가 일어나고 있는지 지속적으로 확인하여 데이터를 수집한다. 또한, 네트워크나 기타 경로를 통해 침입이 들어오고 있는지, 비정상적인 접근이 있는지를 확인하여 데이터를 수집한다. 위와 같은 관련 데이터를 통합하여 데이터 베이스나 파일을 이용하여 저장 및 관리한다. 또한, 사용자가 표준에 맞춰 작성 및 정의한 보안 정책(Security policy)을 인식한다.

둘째 작업은 분석(Analysis)이다. 분석 작업에서는 먼저, 메트릭이나 보안 정책의 유효성을 확인한다. 이를 기반으로 첫째 작업에서 수집 및 통합된 데이터 중 필요한 정보만 필터링한다. 먼저, 필터링한 정보를 기반으로 현재 발생하고 있는 상황, 앞으로 일어날 상황을 예측할 수 있는 모델을 정의한다. 먼저, 상황의 종류 및 가능성을 고려하고, 보안 정책을 기반으로 하여 상황의 심각한 정도 및 우선순위 정보, 정책에 따른 보안 수준 등의 정보를 분석하여 모델에 포함한다. 즉, 앞으로 취해야할 계획을 위한 분석 결과를 위와 같은 모델을 통하여 제공한다.

셋째 작업은 계획(Planning)이다. 계획 작업에서는 먼저, 유효한 보안 정책과 분석 결과를

기반으로 보호를 위한 일련의 액션을 정의한다. 또한, 분석 결과에 따라 액션의 실행 순서를 결정한다.

넷째 작업은 실행(Execution)이다. 계획 단계의 결과를 통해 실제적인 액션을 할 수 있는 방법을 결정하고, 이를 기반으로 실제적인 액션을 취한다. 예를 들어, 예외 처리를 수행하고 침입이 일어나기 전 상태로 다시 복구하는 작업이 일어날 수 있다.

이 외에도 자체보호를 지원하기 위한 메카니즘에는 여러 가지가 있다.

자체보호 속성을 만족하는 아키텍처 명세인지 확인하기 위한 예상 평가 속성으로는 다음과 같은 것이 있다. 보호 대상 시스템에 어떠한 변화가 일어나는지, 어떤 접근이 발생하고 있는지 모니터링 할 수 있는 실시간 감시력, 사용자가 정의한 정책을 인식하고 유효성을 확인할 수 있는 정책 인지력, 신뢰성이 보장된 개체가 접근하는지 신뢰성이 없는 개체가 접근하는지 식별하는 개체 식별력, 결함이 발생할 경우 얼마나 시스템이 견디는지 평가하는 결함 허용성, 신뢰성이 없는 개체의 접근으로 인해 시스템의 손상이 발생했을 경우 이로부터 회복하는 능력을 평가하는 회복력, 자체보호의 능력을 범위를 판단하기 위한 외부 자원 의존도, TSF 보안 기능의 응집력 등이 있다.

제 3 절 Domain Isolation에 대한 해석

영역분리는 보안 아키텍처가 만족해야 하는 3가지 주요 속성 가운데 하나로써 그림 25에서 명시한 바와 같이 보안영역(Security Domain)을 구분해 놓고 신뢰성이 보장되지 않은 개체가 접근할 수 있는 도메인을 지정해 놓고, 신뢰성이 보장되지 않은 개체의 접근으로 인해 다른 영역의 보안성이 낮춰지는 것을 방지하는 속성을 의미한다.

여기서 보안영역을 정의하는 기준, 방법에 대한 이해가 중요하며 선행되어야 한다. 하지만 보안 아키텍처 명세서 제출자나 평가자 모두 보안영역 기준을 정의하는 것에 익숙치 않을 수 있다. 보안영역은 대상 시스템의 업무영역에 따라 달라질 수 있으며 일반적으로 역할(Role)에 따라 분류한다. 예를 들어, 공공기관의 시스템은 그 사용자가 공공기관의 직원이 될 수 있고 일반 국민이 될 수 있다. 이러한 경우, 공공기관 시스템의 보안영역은 공공기관 직원이나 일반 국민이 접근할 수 있는 자원이 된다. 또 다른 예로, 통신사의 경우, 시내 통화, 시외 통화, 국제 통화 등이 보안영역을 나누는 기준이 될 수 있다.

보안 영역을 정의함에 있어서는, 영역의 응집력(cohesion) 및 결합도(coupling)를 고려할 수 있다. 일반적으로 응집력은 높고, 결합도는 낮게 정의해야 한다.

응집력은 보완 관련 작업들이 서로 연관되는 방식과 정도를 의미한다. 응집력의 유형은 우연적(coincidental), 교환적(communicational), 기능적(functional), 논리적(logical), 순차적(sequential), 시간적(temporal) 응집력 등으로 구분된다. 각 응집력의 유형별 특성은 다음과 같고, 바람직한 순서대로 나열하였다.

첫째, 기능적 응집력을 지닌 보안 작업은 보안과 관련된 단일 목적과 관련된 활동을 수행한다. 기능적으로 응집된 보안 작업은 단일 유형의 입력을 단일 유형의 출력으로 변환한다. 둘째, 순차적 응집력을 지닌 보안 작업은 작업 내의 한 보안 기능의 결과가 다른 보안 기능의 입력이 되는 특성을 포함한다. 셋째, 교환적 응집력을 지닌 보안 작업은 작업 내의 다른 기능을 위해 결과를 생성하거나, 다른 보안 기능의 결과를 이용하는 기능을 포함한다. 넷째, 시간적 응집력을 지닌 보안 작업은 거의 동시에 실행되어야 하는 기능들을 포함한다. 예로는 초기화, 복구, 종료 보안 작업을 들 수 있다. 다섯째, 논리적 응집력 혹은 절차적 응집력을 지닌 보안 작업은 서로 다른 데이터 구조에서 유사한 활동을 수행한다. 보안 작업의 기능이 서로 다른 입력에 대하여 서로 다른 오퍼레이션을 수행하는 경우 그 작업은 논리적 응집력 상태이다. 여섯째, 우연적 응집력을 지닌 보안 작업은 서로 관계가 없거나 긴밀하지 않은 여러 활동을 수행한다.

결합도는 보안 작업 간 상호 의존성의 방식과 정도를 의미한다. 결합도의 유형은 호출(call), 공통(common), 내용(content) 결합도 등으로 구분된다. 각 결합도의 유형별 특성은 다음과 같고, 바람직한 순서대로 나열하였다.

첫째, 호출 결합도는 두 작업이 문서화된 보안 기능 호출을 사용해서 엄격하게 통신하는 경우를 의미한다. 호출 결합도에는 데이터, 스탬프(stamp), 제어 등이 있다. 데이터 결합도는 두 작업이 단일 데이터 항목을 나타내는 호출 매개변수를 사용해서 엄격하게 통신하는 경우이다. 스탬프 결합도는 두 작업이 여러 가지 필드를 구성하거나 의미 있는 내부 구조를 갖는 호출 매개 변수를 사용해서 통신하는 경우이다. 제어 결합도는 한 모듈이 다른 모듈의 내부 논리에 영향을 미치게 될 정보를 전송하는 경우이다.

둘째, 공통 결합도는 두 작업이 공통의 데이터 영역이나 공통의 시스템 자원을 공유하는 경우이다. 전역 변수(global variables)를 사용하는 작업은 공통 결합도를 나타낸다고 볼 수 있다. 전역 변수를 통한 공통 결합도는 대부분의 경우 허용되지만 제한된 범위 내에서다. 전역 변수의 적절성을 판단하기 위해 고려해야 할 요소들은 다음과 같다. 먼저, 전역 변수를 변경하는 작업의 수를 고려해야 한다. 일반적으로, 단 한 개의 작업만이 전역 변수의 내용을 제어하는 권한을 부여받아야 한다. 그러나 다른 작업이 그 권한을 공유하게 되는 경우가 있을 수 있는데, 이때는 충분한 정당화가 제공되어야 한다. 다음으로는 전역 변수를 참조하는 작업의 수를 고려해야 한다. 일반적으로 전역 별수를 참조할 수 있는 작업의 수에는 제한이 없지만, 여러 개의 모듈이 전역 변수를 참조할 경우에는 타당성과 필요성 측면이 고려되어야 한다.

셋째, 내용 결합도는 한 작업이 다른 작업의 내부를 직접 참조하는 경우를 의미한다. 즉, 한 작업의 일부 또는 전체 내용이 다른 작업에 실제 포함된다. 내용 결합은 알려지지 않은 작업의 내부 기능을 사용하는 것으로 볼 수 있다. 이는 알려진 작업의 기능만을 사용하는 호출 결합과는 대조적인 것이다.

515 *Domain isolation* is a property whereby the TSF creates separate *security domains* for each untrusted active entity to operate on its resources, and then keeps those domains separated from one another so that no entity can run in the domain of any other. For example, an operating system TOE supplies a domain (address space, per-process environment variables) for each process associated with untrusted entities).

516 For some TOEs such domains do not exist because all of the actions of the untrusted entities are brokered by the TSF. A packet-filter firewall is an example of such a TOE, where there are no untrusted entity domains; there are only data structures maintained by the TSF. The existence of domains, then, is dependant upon 1) the type of TOE and 2) the SFRs levied on the TOE. In the cases where the TOE does provide domains for untrusted entities, this family requires that those domains are isolated from one another such that untrusted entities in one domain are prevented from tampering (affecting without brokering by the TSF) from another untrusted entity's domain.

(그림 25) Domain Isolation이 인용된 CC 원문

특별히 소프트웨어 아키텍처에서는 영역을 구분하는 기준이 아키텍처 설계시 적용한 아키텍처 스타일과 관계가 높다. 소프트웨어 아키텍처 스타일에 적용된 영역분리 장치들로는 관심의 분리, 계층화(Layering), 분할(Partitioning) 등이 있다. 또한, 각 아키텍처 스타일의 특성(Property) 고려하여 영역을 구분할 수 있다. 예를 들어, Publish-Subscribe 스타일에서는 어떠한 정보를 쓰기만하는 컴포넌트, 쓰여진 정보를 읽기만 하는 컴포넌트로 구분될 수 있기 때문에 읽고 쓰는 역할에 따라 영역을 분리할 수 있다.

영역분리 속성을 만족하는 아키텍처 명세인지 확인하기 위한 주요 평가 속성으로는 다음과 같은 속성이 있다. 보안 영역을 구분하는 기준이 타당한지, 구분에 기준에 따라 적절하게 구분되었는지를 평가하는 영역 구별 적절성, 영역 구별을 검증하기 위해 영역 간에 의존성은 높지 않은지 평가하는 영역 간 의존성과 영역 내 응집력이 그것이다. 영역 간에 의존성이 높아지면 신뢰되지 않은 개체의 접근으로 인해 주변의 영역에 그 영향이 미칠 수 있다.

제 4 절 Non-bypassability에 대한 해석

우회불가성은 보안 아키텍처가 만족해야 하는 3가지 주요 속성 가운데 하나로써 그림 26와 같이 TOE가 제공하는 모든 기능이 TSF의 보안 기능을 우회하지 않고 모두 거쳐가는 속성을 의미한다.

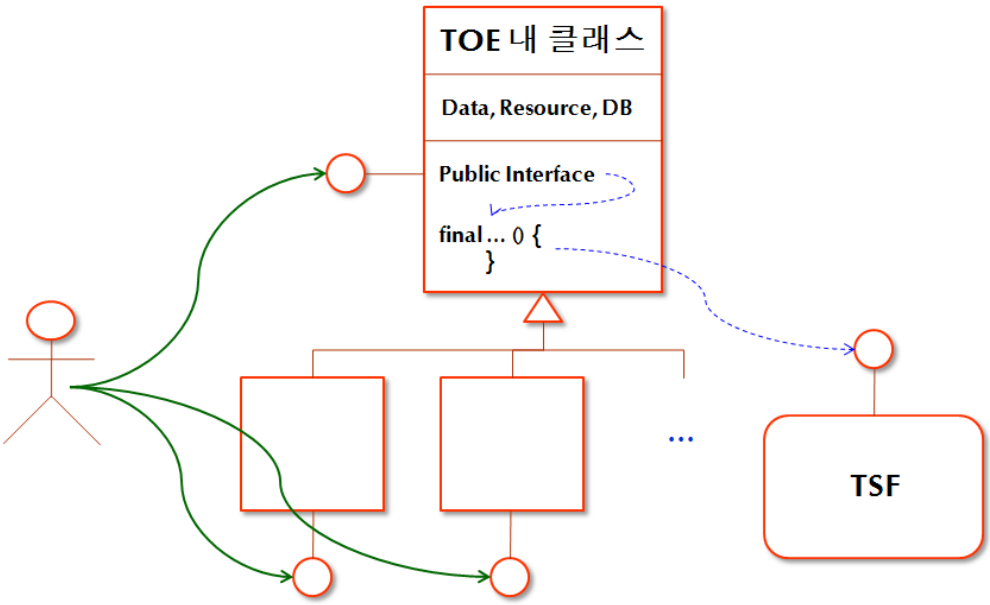
517 *Non-bypassability* is a property that the security functionality of the TSF (as specified by the SFRs) is always invoked and cannot be circumvented when appropriate for that specific mechanism. For example, if access control to files is specified as a capability of the TSF via an SFR, there must be no interfaces through which files can be accessed without invoking the TSF's access control mechanism (an interface through which a raw disk access takes place might be an example of such an interface).

518 As is the case with self-protection, the very nature of some TOEs might depend upon their environments to play a role in non-bypassability of the TSF. For example, a security application TOE requires that it be invoked by the underlying operating system. Similarly, a firewall depends upon the fact that there are no direct connections between the internal and external networks and that all traffic between them must go through the firewall.

(그림 26) Non-bypassability가 인용된 CC 원문

우회불가성을 적용한 아키텍처 설계를 위해서는 TSF를 TOE에 접근하기 위한 유일한 통로(Gateway)로 제한하는 것이 좋다. 즉, TSF의 인터페이스를 통해서만 TOE의 인터페이스를 접근하게 하는 것이다. 또는 TSF의 인터페이스와 TOE의 인터페이스 간에 검증 동기화(Validation Synchronization)를 함으로써 TOE의 기능을 사용하기 위해서는 필히 TSF의 기능을 사용하게 한다. 검증 동기화를 위한 메커니즘으로는 핸드 쉐이킹(Hand-shaking), 암호화 등이 있다.

객체지향 기반의 자바를 이용한 설계 시에는 그림 27과 같이 TOE 내의 클래스에 자바 언어의 'Final' 같은 장치로 TSF의 기능을 접근하는 부분을 정의하고 TOE의 클래스를 sub-classing 함으로써 TSF 기능을 사용하는 부분이 포함된 인터페이스를 반드시 호출하게 한다. 이러한 설계를 이용할 경우, TSF의 기능을 접근하는 부분에 대한 상속을 통해 우회 접근을 차단한다.



(그림 27) 우회불가성 설계의 예

우회불가성을 평가하기 위해서는 수학적인 메트릭을 통한 정량적인 방법으로는 평가가 힘들며 우회불가성을 지원하기 위한 설계서를 보고 설계 내용이 우회불가성을 만족하기 위한 기준들을 준수했다는 것을 판단함으로 평가될 수 있다.

제 5 절 보안 아키텍처 명세서 (Security Architecture Description)

CC 스팩에서는 보안 아키텍처 명세서가 기술하고 있어야 하는 정보에 대해 명시하고 있다. 명세서 내에는 특별히 TSF가 어떻게 영역분리, 자체보호, 우회불가성을 지원하는지에 관해 명세하고 있어야 한다. CC에서 언급하고 있는 필요한 정보들은 향후 아키텍처 명세 지침 및 평가 지침을 개발하기 위한 근거로 활용될 수 있다.

1. 보안 아키텍처 명세서의 개괄적인 활용

본 섹션에서는 CC 본문의 내용을 기반으로 보안 아키텍처 명세서 작성시에 고려해야 하는 사항과 평가시에 고려해야 하는 사항을 추출한다. 추출된 사항들은 명세서 작성 지침과 평가 지침 개발시에 활용된다.

519 The security architecture description explains how the properties described above are exhibited by the TSF. It describes how domains are defined and how the TSF keeps them separate. It describes what prevents untrusted processes from getting to the TSF and modifying it. It describes what ensures that all resources under the TSF's control are adequately protected and that all actions related to the SFRs are mediated by the TSF. It explains any role the environment plays in any of these (e.g. presuming it gets correctly invoked by its underlying environment, how are its security functions invoked?).

520 The security architecture description presents the TSF's properties of self-protection, domain isolation, and non-bypassability in terms of the decomposition descriptions. The level of this description is commensurate with the TSF description required by the ADV_FSP, ADV_TDS and ADV_IMP requirements that are being claimed. For example, if ADV_FSP is the only TSF description available, it would be difficult to provide any meaningful architectural design because none of the details of any internal workings of the TSF would be available.

(그림 28) 보안 아키텍처 명세서 관련 CC 본문 - 519, 520

보안 아키텍처 명세서에는 TSF가 어떻게 자체보호, 영역분리, 우회불가성을 지원하는지 명시적으로 기술하는 항목, 보안 아키텍처 내 영역의 종류를 나누는 기준을 기술한 항목, 영역 구분 기준을 적용하여 영역을 분리한 방법에 대해 기술한 항목, 영역간 독립성이 보장됨을 보여주는 항목, 신뢰성이 보장되지 않은 프로세스의 접근으로 TSF의 수정을 막는 방법을 기술한 항목이 명시되어야 한다. 또한, TSF가 제어하고 있는 모든 리소스가 침입으로부터 적절하게 보호되고 있다는 것과 모든 SFR 요구사항이 TSF에 의해 중재되고 있음을 보여주어야 한다. TSF가 3 가지 보안 특성이 만족되고 있다는 것을 보여주기 위해 체계적인 항목으로 기술되어 있어야 하며 기술 상세도는 SFR을 설계하고 구현하기 위해 제공된 명세서의 기술 정도와 비슷하다.

위에서 제시한 작성 지침을 위한 활용 내용은 보안 아키텍처 명세서 평가를 위한 지침 개발시에 활용될 수 있다. 아키텍처 명세서에 자체보호, 영역분리, 우회불가성 지원에 관해 기술한 항목이 존재하는지, 체계적인 항목으로 기술되었는지, 기술 상세도는 평가하기에 적절한지 등을 판단해야 한다. 보안 아키텍처 내 영역의 종류를 나누는 기준을 기술한 항목이 존재하는지, 그 기준이 합당한지, 그 기준에 의해 적절하게 분할이 되어서 영역간 독립성이 보장되고 있는지 확인해야 한다. 신뢰성이 보장되지 않은 개체의 정의는 무엇이며, 그러한 개체로부터 TSF를 함부로 접근하는 것을 어떻게 방지하는지, 방지하는 방법은 적절한지 등을 확인해야 한다. 또한, TSF가 제어하는 리소스들이 어디까지이며, 침입으로부터 제대로 보호되는지 평가해야 한다.

521	However, if the TOE design were also available, even at the most basic level (ADV_TDS.1), there would be some information available concerning the subsystems that make up the TSF, and there would be a description of how they work to implement self-protection, domain isolation, and non-bypassability. For example, perhaps all user interaction with the TOE is constrained through a process that acts on that user's behalf, adopting all of the user's security attributes; the architectural design would describe how such a process comes into being, how the process's behaviour is constrained by the TSF (so it cannot corrupt the TSF), how all actions of that process are mediated by the TSF (thereby explaining why the TSF cannot be bypassed), etc.

(그림 29) 보안 아키텍처 명세서 관련 CC 본문 - 521

보안 아키텍처 명세서 작성시에는 TSF를 구성하는 서브시스템에 관해 기술한 항목이 있어야 하며, 각 서브시스템이 자체보호, 영역분리, 우회불가성 특성을 어떻게 구현하는지 기술해야 한다.

앞에서 도출한 작성 항목을 기반으로 보안 아키텍처 명세서 평가 지침을 도출할 수 있다. 명세서 내에 TSF를 구성하는 서브시스템을 기술한 항목이 있는지, 적절한 수준에서 기술되었는지 평가해야 하며, 서브시스템이 3 가지 특성을 어떻게 구현하는지 기술한 항목이 존재해야 하고, 지원 여부를 평가할 수 있을 정도의 상세도로 기술되어야 한다.

522	If the available TOE design is more detailed (e.g. at the modular level), or the implementation representation is also available, then the description of the architectural design would be correspondingly more detailed, explaining how the user's process communicate with the TSF processes, how different requests are processed by the TSF, what parameters are passed, what programmatic protections (buffer overflow prevention, are in place, parameter bounds checking, time of check/time of use checking), etc. Similarly, a TOE whose ST claimed the ADV_IMP component would go into implementation-specific detail.
523	The explanations provided in the security architecture description are expected to be of sufficient detail that one would be able to test their accuracy. That is, simple assertions (e.g. "The TSF keeps domains separate") provide no useful information to convince the reader that the TSF does indeed create and separate domains.

(그림 30) 보안 아키텍처 명세서 관련 CC 본문 - 522, 523

보안 아키텍처 명세서 작성시에는 TSF과 연계해서 실행될 수 있는 사용자의 프로세스에는 무엇이 있으며, 사용자의 프로세스가 TSF의 프로세스와 어떻게 상호 작용하는지, 얼마나 다양한 종류의 프로세스들이 TSF에 의해 어떻게 처리될 수 있는지 기술해야 한다. 또한, TSF에서 입력 받는 수 있는 파라미터는 무엇인지, 보호 기능을 어떻게 구현하는지 등에 대해 기술해야 한다.

명세서 작성시 고려해야 하는 내용을 기반으로 평가 지침을 도출할 수 있다. 사용자의 프로세스가 TSF의 프로세스와 어떻게 상호 작용하는지 기술한 항목이 존재해야 하며, 다른 종류의 프로세스들이 TSF에 의해 어떻게 처리될 수 있는지 기술하는 항목이 있어야 한다. 또한, 처리할 수 있는 파라미터에 대해 기술한 항목이 있어야 하고, 보호 기능을 지원하기 위한 프로그래밍적 지원에 대해 기술한 항목이 있어야 한다. 여기서 중요한 것은 각 항목이 단순히 존재만 해서는 안된다는 것이다. CC 본문에서 기술한 바와 같이, 위의 항목이 적절히 기술되었는지 평가자가 평가할 수 있을 정도의 충분한 상세도로 기술이 되어야 한다.

2. 영역분리 명세

보안 아키텍처 명세서 내에 영역분리 특성 지원에 관해 명세한 항목이 있어야 한다. 본 섹션에서는 CC 본문의 내용을 기반으로 영역분리 관련 명세서 작성시에 고려해야 하는 사항과 평가시에 고려해야 하는 사항을 추출한다. 추출된 사항들은 명세서 작성 지침과 평가 지침 개발시에 활용된다.

524	In cases where the TOE exhibits domain isolation entirely on its own, there would be a straightforward description of how this is attained. The security architecture description would explain the different kinds of domains that are defined by the TSF, how they are defined (i.e. what resources are allocated to each domain), how no resources are left unprotected, and how the domains are kept separated so that active entities in one domain cannot tamper with resources in another domain.
525	For cases where the TOE depends upon other IT entities to play a role in domain isolation, that sharing of roles must be made clear. For example, a TOE that is solely application software relies upon the underlying operating system to correctly instantiate the domains that the TOE defines; if the TOE defines separate processing space, memory space, etc, for each domain, it depends upon the underlying operating system to operate correctly and benignly (e.g. allow the process to execute only in the execution space that is requested by the TOE software).

(그림 31) 영역분리 명세 관련 CC 본문 - 524, 525

TSF가 영역분리 특성을 지원한다는 것을 보여주기 다음과 같은 항목을 기술해야 한다. TSF 내에 정의된 영역의 종류, 영역을 분류한 기준, 영역에 할당된 자원, 모든 자원이 보호되고 있다는 것에 대한 근거, 영역간에 독립성이 보장되는 이유 등에 대해 기술해야 한다.

작성 항목을 기반으로 평가를 위한 활용 내용을 도출하면 다음과 같다. TSF 내에 정의된 영역의 종류를 충분히 기술한 항목이 있는지, 영역 분류 기준을 기술한 항목이 있는지, 영역에 할당된 자원을 기술한 항목이 있는지, 모든 자원이 보호되고 있다는 것에 대한 근거를 기술한 항목이 있는지, 영역간에 독립성이 보장되는 이유를 기술한 항목이 있는지 확인해야

한다. 또한, 이러한 항목들이 충분한 상세도로 기술되었는지 평가해야 한다.

526 For example, mechanisms that implement domain separation (e.g., memory management, protected processing modes provided by the hardware, etc.) would be identified and described. Or, the TSF might implement software protection constructs or coding conventions that contribute to implementing isolation of software domains, perhaps by delineating user address space from system address space.

(그림 32) 영역분리 명세 관련 CC 본문 - 526

영역분리 관련 명세에 추가적으로 영역 분리를 구현하는 메커니즘 및 TSF에서 소프트웨어적으로 보호 기능을 구현한 것에 대해 명세해야 한다. 그리고, 평가시에는 영역 분리를 구현하는 메커니즘을 기술한 항목이 있는지, 메커니즘이 적절한지, TSF에서 소프트웨어적으로 보호 기능을 구현한 것에 대해 기술한 항목이 있는지, 구현 내용이 유요한지 평가해야 한다.

3. 자체보호 명세

보안 아키텍처 명세서 내에 자체보호 특성 지원에 관해 명세한 항목이 있어야 한다. 본 섹션에서는 CC 본문의 내용을 기반으로 자체보호 관련 명세서 작성시에 고려해야 하는 사항과 평가시에 고려해야 하는 사항을 추출한다. 추출된 사항들은 명세서 작성 지침과 평가 지침 개발시에 활용된다.

528 In cases where the TOE exhibits self-protection entirely on its own, there would be a straightforward description of how this self-protection is attained. Mechanisms that use domain separation to define a TSF domain that is protected from other (user) domains would be identified and described.

529 For cases where the TOE depends upon other IT entities to play a role in protecting itself, that sharing of roles must be made clear. For example, a TOE that is solely application software relies upon the underlying operating system to operate correctly and benignly; the application cannot protect itself against a malicious operating system that subverts it (for example, by overwriting its executable code or TSF data).

(그림 33) 자체보호 명세 관련 CC 본문 - 528, 529

TSF가 자체보호 특성을 지원한다는 것을 보여주기 위해 TSF에서 영역을 구분하기 위해 사용한 메커니즘에 대해 기술해야 한다. 또한, 자체 보호 기능을 제공하기 위해 외부의 IT

개체를 사용할 경우 역할 구분에 대해 작성해야 한다.

평가시에는 TSF 영역을 정의하기 위해 사용한 메커니즘이 정의되었는지, 합당한지 식별해야 하며, 자체 보호 기능을 제공하기 위해 외부의 IT 개체를 사용할 경우 역할 구분에 대해 기술하였는지 평가해야 한다.

530	The security architecture description also covers how user input is handled by the TSF in such a way that the TSF does not subject itself to being corrupted by that user input. For example, the TSF might implement the notion of privilege and protect itself by using privileged-mode routines to handle user data. The TSF might make use of processor-based separation mechanisms (e.g. privilege levels or rings) to separate TSF code and data from user code and data. The TSF might implement software protection constructs or coding conventions that contribute to implementing isolation of software, perhaps by delineating user address space from system address space.

(그림 34) 자체보호 명세 관련 CC 본문 - 530

자체보호 관련 명세서 작성시에는 사용자 입력을 처리하는 방식에 대해 기술해야 하며, 사용자 입력에 의해 TSF의 보안성이 해쳐지지 않음을 보여주어야 한다. 사용자 입력을 처리하는 방법으로는 실행되는 기능에 따라 특혜 (privilege) 수준을 부여하고 특혜 수준에 따라 사용자 입력을 처리하는 방식이 있다.

평가시에는 TSF가 받는 사용자 입력에 대해 기술하고 있는지, TSF의 보안성을 해치도록 유도하는 사용자 입력을 구별하고 있는지 확인해야 한다. 또한, 사용자 입력에 대해 처리하는 방식이 적절한지 평가해야 한다.

531	For TOEs that start up in a low-function mode (for example, a single-user mode accessible only to installers or administrators) and then transition to the evaluated secure configuration (a mode whereby untrusted users are able to login and use the services and resources of the TOE), the security architecture description also includes an explanation of how the TSF is protected against this initialisation code that does not run in the evaluated configuration. For such TOEs, the security architecture description would explain what prevents those services that should be available only during initialisation (e.g. direct access to resources) from being accessible in the evaluated configuration. It would also explain what prevents initialisation code from running while the TOE is in the evaluated configuration.
532	There must also be an explanation of how the trusted initialisation code will maintain the integrity of the TSF (and of its initialisation process) such that the initialisation process is able to detect any modification that would result in the TSF being spoofed into believe it was in an initial secure state.

(그림 35) 자체보호 명세 관련 CC 본문 - 531, 532

명세서 작성시에는 또한 TOE 시작 후 자체 보호 모드로 들어가기 전에 단일 사용자 모드에 머물고 있을 때, 초기화를 위해 실행되는 초기화 코드는 무엇인지 명시해야 하며 초기화 코드로부터 어떻게 TSF가 보호되는지 명시해야 한다. 초기화 코드가 실행되고 있을 때, 사용 가능한 기능과 사용 가능하지 않은 기능에 대해서도 구분해서 기술해야 하며 자체 보호 모드로 들어갔을 경우, 실행되지 않아야 하는 초기화 코드는 무엇인지 기술해야 한다. 더 나아가 신뢰성이 보장된 초기화 코드는 무엇이며 TSF의 무결성을 어떻게 보장해주는지 기술해야 한다.

평가시에는 초기화 코드 실행 모드와 자체 보호 모드의 구분에 대해 기술하였는지, 실행되는 초기화 코드에 대해 충분히 기술하였는지 확인해야 하며 초기화 코드로부터 어떻게 TSF가 보호되는지에 대해 충분히 기술하였는지 평가해야 한다. 초기화 코드가 실행될 때, 사용 가능한 기능과 사용 가능하지 않은 기능에 대해 명시하였는지, 자체 보호 모드로 들어갔을 경우, 실행되지 않아야 하는 초기화 코드는 무엇인지 명시하였는지 확인해야 한다. 또한, 신뢰성이 보장된 초기화 코드가 TSF의 무결성을 보장해줌을 적절히 기술되었는지 평가해야 한다.

4. 우회불가성 명세

보안 아키텍처 명세서 내에 우회불가성 특성 지원에 관해 명세한 항목이 있어야 한다. 본 섹션에서는 CC 본문의 내용을 기반으로 우회불가성 관련 명세서 작성시에 고려해야 하는 사항과 평가시에 고려해야 하는 사항을 추출한다. 추출된 사항들은 명세서 작성 지침과 평가 지침 개발시에 활용된다.

534 The property of non-bypassability is concerned with interfaces that permit the bypass of the enforcement mechanisms. In most cases this is a consequence of the implementation, where if a programmer is writing an interface that accesses or manipulates an object, it is that programmer's responsibility to use interfaces that are part of the SFR enforcement mechanism for the object and not to try to circumvent those interfaces. For the description pertaining to non-bypassability, then, there are two broad areas that have to be covered.

(그림 36) 우회불가성 명세 관련 CC 본문 - 534

TSF가 우회불가성 특성을 지원함을 보여주기 위해 다음과 같은 항목을 기술해야 한다. SFR 보장을 위해 반드시 호출해야 하는 인터페이스가 무엇인지 기술해야 하며 TOE의 인터페이스가 그 기능을 우회하지 않고 반드시 사용하였다는 것을 기술해야 한다. 그리고 2 가지 기술해야 하는 정보에 관해 그림 37과 그림 38에서 다루고 있다. 하나는 SFR-수행과 관

련한 인터페이스에 관한 설명이고 다른 하나는 SFR-수행과 관련되지 않은 인터페이스에 관한 설명이다.

평가시에는 SFR 보장을 위해 반드시 호출해야하는 인터페이스에 대해 명확히 기술하였는지 확인해야 하며 TOE 기능이 그 인터페이스를 호출하였다는 것을 충분한 상세도로 기술했는지 확인해야 한다. 또한, SFR-수행 관련/비관련 인터페이스에 대해 충분히 기술했는지 평가해야 한다.

535 The first consists of those interfaces to the SFR-enforcement. The property for these interfaces is that they contain no operations or modes that allow them to be used to bypass the TSF. It is likely that the evidence for ADV_FSP and ADV_TDS can be used in large part to make this determination. Because non-bypassability is the concern, if only certain operations available through these TSFIs are documented (because they are SFR-enforcing) and others are not, the developer should consider whether additional information (to that presented in ADV_FSP and ADV_TDS) is necessary to make a determination that the non-SFR-enforcing operations of the TSFI do not afford an untrusted entity the ability to bypass the policy being enforced. If such information is necessary, it is included in the architectural design document.

(그림 37) 우회불가성 명세 관련 CC 본문 - 535

우회불가성 관련 명세서 중 첫 번째 항목인 SFR-수행 관련 인터페이스 명세서 작성시에는 인터페이스 내의 오퍼레이션의 범위, 오퍼레이션의 시그너쳐에 대해 기술해야 하며, 이러한 인터페이스가 어떠한 실행 모드에서도 우회되지 않는 근거에 대해 기술해야 한다.

평가시에는 SFR-수행 관련 인터페이스 내의 오퍼레이션에 대한 설명이 충분히 기술되었는지, 실행 모드 중에 TSF의 인터페이스를 우회하는 모드가 없다는 것에 대한 근거가 충분히 기술되었는지 평가해야 한다.

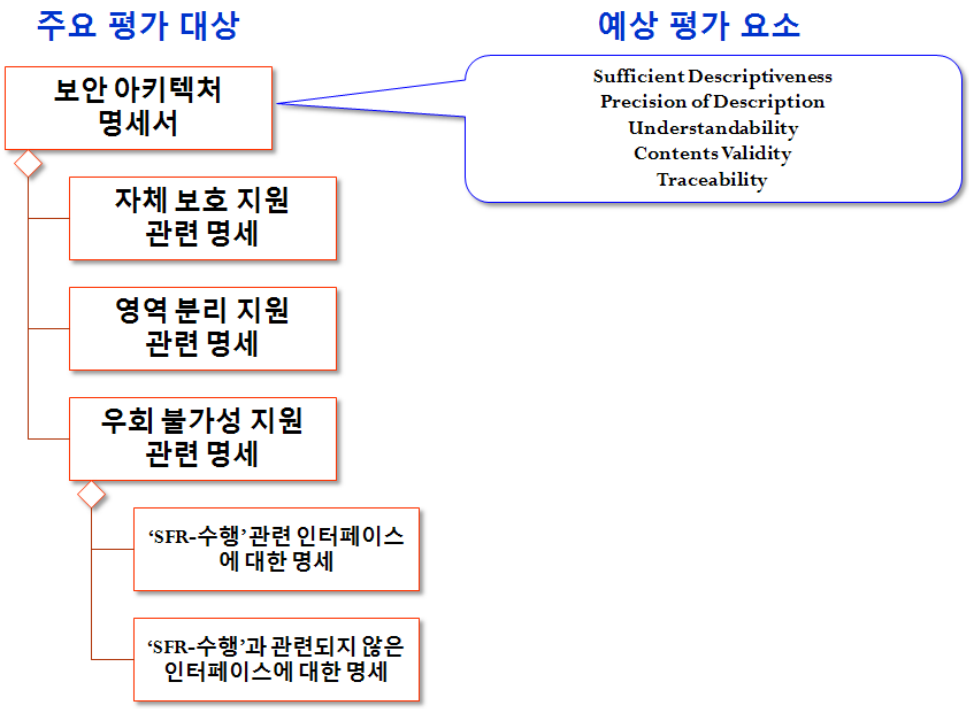
536 The second area of non-bypassability is concerned with those interfaces whose interactions are not associated with SFR-enforcement. Depending on the ADV_FSP and ADV_TDS components claimed, some information about these interfaces may or may not exist in the functional specification and TOE design documentation. The information presented for such interfaces (or groups of interfaces) should be sufficient so that a reader can make a determination (at the level of detail commensurate with the rest of the evidence supplied in the ADV: Development class) that the enforcement mechanisms cannot be bypassed.

(그림 38) 우회불가성 명세 관련 CC 본문 - 536

우회불가성 관련 명세서 중 두 번째 항목인 SFR-수행과 관련되지 않은 인터페이스 명세

서 작성시에는 첫 번째 항목과 비슷하게 인터페이스 내 오퍼레이션의 범위, 시그너쳐에 대해 기술해야 한다. 그리고 평가시에는 SFR-수행과 관련되지 않은 인터페이스 내의 오퍼레이션 중에 TSF를 우회하는 것은 없는지 평가해야 한다.

앞에서 도출한 작성 지침 및 평가 지침 반영 내용을 기반으로 주요 평가 대상 및 예상 평가 요소를 도출하면 그림 39과 같다. 주요 평가 대상은 보안 아키텍처 명세서 내의 자체 보호, 영역분리, 우회불가성 지원 관련 명세이며, 명세서를 평가하기 때문에 공통적으로 적용될 수 있는 평가 요소에는 명세서 내에 기술이 충분히 되었는지 평가하는 Sufficient Descriptiveness, 기술된 내용이 충분한 상세도로 기술되었는지 평가하는 Precision of Description, 기술된 내용이 이해하기 쉬운지 평가하는 Understandability, 기술된 내용이 올바르고 유효한지 평가하는 Contents Validity, 기술된 항목 간에 일관성이 깨지지 않고 추적이 되는지 평가하는 Traceability 등이 있다.



(그림 39) 주요 평가 대상 및 예상 평가 요소

제 5 장 TOE 설계 (ADV_TDS)의 소프트웨어 공학적 분석

ADV_TDS는 보안기능 요구사항(SFR)을 어떻게 설계하였는지 서술하기 위한 요구사항을 다양한 추상화 수준으로 설명한다. 이 패밀리는 TOE 개발 과정에서 SFR을 완전하고 정확하게 설계하였는지를 입증하며, 설계와 구현의 일치성을 확인하기 사용된다. 본 장에서는 ADV_TDS에 기술된 내용 소프트웨어 공학적으로 분석한다.

제 1 절 설계 명세서의 충분성 (Sufficiency)

TOE의 설계 명세서(design description)는 TSF 명세를 위한 컨텍스트 정보와 TSF의 완전한 명세를 포함하며 TOE 개발 과정에서 SFR에 대한 설계 내용을 기술한다 [5]. 높은 수준의 보증(assurance)을 요구함에 따라 설계 명세서에 포함되어야 하는 내용의 상세도도 높아진다.

그림 40와 같이 설계 명세서의 목적은 TSF의 경계를 결정하고 TSF가 보안기능 요구사항(SFR)이 어떻게 설계 또는 구현되는지 서술하기 위한 충분한 정보를 제공하는 것으로, 설계 명세서의 분량 및 구조는 TOE의 복잡도와 크기에 따라 다르다.

275 The goal of design documentation is to provide sufficient information to determine the TSF boundary, and to describe *how* the TSF implements the Security Functional Requirements. The amount and structure of the design documentation will depend on the complexity of the TOE and the number of SFRs; in general, a very complex TOE with a large number of SFRs will

(그림 40) 설계 명세서의 목적이 명시된 CC 원문

그러나, 소프트웨어 공학에서는 어느 정도의 정보를 기술해야 충분한 것인지를 결정하는 것이 힘들다. 즉, 설계 명세서의 내용은 비즈니스 도메인 성격, 프로젝트의 규모(비용, 기간 측면 고려), 소프트웨어의 복잡도와 크기, 사용되는 방법론(절차적 방법론, 객체지향 방법론, 컴포넌트 기반 개발 방법론 등)에 따라서 다르기 때문에, 문서의 구조나 분량 측면에서 충분한지 불충분하게 기술되었는지 정량적으로 측정하기 힘들다. 게다가 충분한 정도의 정보에 의미해서 명세서가 정확하게 기술되어야 한다는 의미를 포함한다. 그러므로 소프트웨어 관점에서는 "sufficient information"을 결정하는 것은 매우 어려운 작업이다.

정량적인 방법은 아니지만, 정성적으로 결정할 수 있는 기준은 제시할 수 있다. 비교적 쉬운 접근 방법은 템플릿과 체크리스트를 이용하는 것으로, 개발자가 제출해야 하는 명세서의 템플릿을 미리 정의하고 이를 평가하기 위한 항목을 포함하는 체크리스트를 만든다. 먼저, 평가자는 대표적인 TSF 목록을 이미 식별하고, 각 TSF가 어떻게 구현될 수 있는지를

미리 알고 있으면 이를 기반으로 필요한 산출물을 도출할 수 있고, 각 산출물에 충분한 정보가 포함되도록 템플릿을 만든다. 그리고, 템플릿을 기반으로 만든 명세서의 완전성을 평가할 수 있는 광범위한 범위로 정의된 체크리스트를 이용하여 명세서를 평가한다.

또는, 소프트웨어 설계자가 설계시 고려한 TSF의 항목들을 나열하고, 이 TSF들이 어떻게 설계되었는지를 증명함으로써 설계서의 충분성을 입증할 수 있다. 한 방법으로는 소프트웨어 테스팅 기법을 이용하는 것이다. 소프트웨어 테스팅 기법을 이용하여 고려한 TSF를 위한 테스트 케이스를 구축하고, 테스트 케이스와 관련된 산출물과 소스코드를 테스트함으로써 설계서의 충분성을 입증할 수 있다. 또 다른 방법으로는 Architecture Soundeness를 입증하는 것과 같이 추적성(Traceability)를 이용할 수 있다. 해당 소스 코드로부터 추적하여 소스 코드와 관련 있는 설계서 또는 분석서를 찾아, 코드에 있는 내용이 충분히 설계서에 작성되어 있는지를 확인한다. 그리고, 해당 설계서가 나타내고 있는 TSF를 역으로 찾아 설계자가 고려한 모든 SFR가 설계서 또는 소스코드에 실체화(realize)되어 있는지를 확인할 수 있다.

충분성에는 기준이 없기 때문에, 템플릿과 체크리스트과 같은 방법을 이용하는 것은 충분한 정보의 의미를 제한할 수 있는 하나의 수단이 되지만 이 역시 템플릿과 체크리스트를 만든 평가자의 능력에 좌우되고 체크리스트의 각 항목에 대한 평가는 평가자의 의견에 의존적으로 결정된다.

제 2 절 Formalism에 대한 해석

본 절에서는 3가지 단계의 Formalism인 비정형화된 명세, 준정형화된 명세, 정형화된 명세에 대해서 소프트웨어 공학 관점에서 해석한다. 각 항목은 3가지 단계의 명세 수준의 장단점과 함께 명세서를 기술하는 방법과 평가하는 방법을 설명한다.

1. 비정형화된 명세 (Informal Specification)

비정형화된 명세는 자연어(한국어, 영어, 중국어 등 일반적으로 사용되는 언어)를 이용하여 기술한 것으로, 기본적으로 필요한 문법이나 구문 등을 제외하고는 명세하기 위한 요구사항이나 제한사항은 없다. 설계 명세서를 작성하기 위한 표기상의 요구사항이나 제약사항이 없기 때문에 개발자가 설계 내용을 기술할 수 있는 표현력은 높아진다. 그러나 평가자의 입장에서는 비정형화된 명세는 평가하기 힘들다. 왜냐하면, 명세서를 작성하기 위한 요구사항이나 제한사항이 없는 것은 이를 평가하기 위한 평가항목이 확정되지 않는다는 것을 의미하기 때문이다. 비정형 방법으로 기술하면 같은 내용일지라도 개발자에 따라서 다르게 기술될 것이고, 이를 동일한 기준을 가지고 평가하는 것은 평가자의 주관에 많이 의존하게 된

다.

[표 2] 비정형화된 명세의 장/단점

장점	단점
· 명세의 제약사항이 없어 개발자의 표현력이 높아짐.	· 애매모호성이 증가함.

개발자는 비정형화 표현기법을 이용하여 설계 명세서를 작성할 경우, 각 SFR이 어떻게 설계되었는지 구체적으로 설명하고 설계에 대한 근거를 합당하게 제시해야 한다. 평가자는 개발자가 제시한 명세서가 모든 SFR 설계를 포함하고 있는지 설계 근거를 이용하여 평가한다. 평가자가 비정형화된 문서를 평가하는 것을 보다 용이하고 공정하게 평가하기 위해서 미리 정의한 평가 항목을 포함하는 체크리스트를 이용할 수 있다. 체크리스트의 평가 항목은 설계 문서의 여러 측면을 평가할 수 있도록 정의되어야 하지만, 비정형화된 문서를 대상으로 평가 항목을 도출하는 것 역시 매우 어려운 작업이다.

2. 준정형화된 명세 (Semi-formal Specification)

준정형화된 명세는 비정형화된 명세와 같이 대부분 자연어를 사용하여 기술하지만, 문서의 형식과 표현 방식이 다르다. 비정형화된 명세의 모호성을 줄이기 위하여 표준 템플릿을 사용하고, 명세서에 사용되는 용어를 미리 정의하며, 다이어그램 또는 간단한 수학적인 표현 방법을 이용하여 기술한다.

준정형화 표현 기법을 이용하여 명세서를 작성하게 되면 비정형화된 명세의 모호성을 줄이는 장점이 있는 반면에 개발자가 설계 내용을 기술하는데 표현력이 줄어들 수 있다. 즉, 준정형화된 명세에는 기술 규칙이나 제약사항 등이 정해져 있기 때문에 자연어로 표현할 때보다 개발자가 문서를 작성하기 어렵게 된다. 준정형화된 표현 방법의 예로서는 UML 다이어그램과 이를 설명하는 텍스트 명세를 포함하는 경우이다.

[표 3] 준정형화된 명세의 장/단점

장점	단점
· 명세 규칙이 정해져 있기 때문에 문서의 애매모호성이 감소함.	· 개발자의 표현력이 감소함. (명세를 위해 다이어그램 표기법이나 수학 표현 방법 등을 알아야 함)

개발자는 준정형화 표현기법을 이용하여 설계 명세서를 작성할 경우, 각 SFR이 어떻게 설계되었는지는 UML의 여러 다이어그램이나 간단한 수학적 표현을 통해서 설명할 수 있

으며, 설계에 대한 근거를 텍스트로 설명함으로써 제시할 수 있다. 평가자는 다이어그램, 수학 표현, 텍스트 설명 등을 기반으로 모든 SFR 설계를 포함하고 있는지를 비정형화된 명세서보다 쉽고 명확하게 평가할 수 있게 된다. 비정형화된 명세와 마찬가지로 준정형화된 명세를 평가하기 위해서 체크리스트를 이용할 수 있다. 그리고, 준정형화된 명세에는 규칙이 있기 때문에 이를 기반으로 평가 항목을 도출하는 것은 비정형화된 명세보다 쉽다. 예를 들어, 올바른 표기법을 사용하였는지, 개발자가 설계 의도에 맞게 표기법을 사용하였는지가 명세 규칙에 기반하여 도출한 평가 항목이 될 수 있다.

3. 정형화된 명세 (Formal Specification)

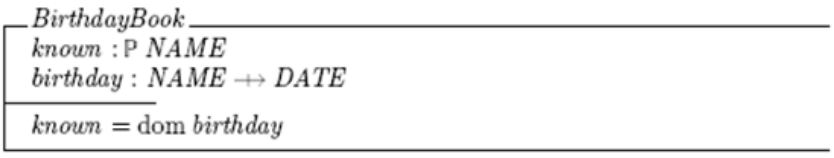
정형화된 명세는 잘 정립된 수학적인 개념을 이용하여 기술하며, 이런 수학 개념은 잘 정의된 Semantic 과 Syntax, 추론 규칙을 정의하기 위해 사용된다. 정형화된 명세는 대부분 소프트웨어가 어떻게 기능을 수행하는지 보다는 소프트웨어의 기능이 무엇인지에 관한 내용을 포함한다.

개발자는 수학 표현기법을 완전히 이해해야 정형화된 문서를 기술할 수 있기 때문에 작성하는데 어려움이 있고, 수학 표현기법을 이용하여 기술할 수 있는 내용은 자연어를 사용할 때보다 제한적이다. 특히 대화형 시스템이나 병행 시스템에는 현재 기술들을 이용하여 정형적인 기법으로 표현하기가 힘들기 때문에 정형 명세는 제한된 경우에 사용된다. 그러나, 기술된 명세서는 수학 개념을 이용하여 표현되기 때문에 모호성이 감소하게 되며, 수학적으로 분석될 수 있으므로 명세서의 일관성과 완전성을 증명하는데 용이하며, 요구사항 명세서와 설계 결과가 일치하는지를 증명할 수 있다.

[표 4] 정형화된 명세의 장/단점

장점	단점
· 수학적인 표현 기법을 사용하므로, 애매모호성이 감소함. · 간결하게 표현 가능 · 명세서의 일관성, 완전성, 일치성을 증명하기 용이함.	· 개발자의 표현력이 감소함. (수학 표현 기법을 알아야 정형화된 명세를 기술할 수 있음) · 이해하기 힘듬. · 적용되는 범위가 한정적임.

정형화된 명세의 예는 Z 명세 언어, ACL2, Isabelle, B Method 등이 있으며, 그림 41은 Z 명세 언어의 간단한 예제를 보여준다.



(그림 41) Z 명세 언어의 예제

개발자는 정형화된 표현기법을 이용하여 설계 명세서를 작성할 경우, 미리 수학 개념을 완전히 습득하고 이해하여 각 SFR이 어떻게 설계되었는지, 설계에 대한 근거를 수학 개념을 통해 기술하고 증명해야 하는 어려움을 가지게 된다. 이에 반해 평가자는 수학 개념을 이용해 각 설계가 정확하게 설계되었는지를 보다 쉽게 증명할 수 있기 때문에 다른 명세보다 용이하게 평가할 수 있게 된다. TOE가 SFR을 정확하게 설계하였는지를 증명하기 위한 formal verification 방법을 사용할 수 있다.

제 3 절 서브시스템과 모듈

ADV_TDS 패밀리는 그림 42과 같이 서브시스템과 모듈이라는 두 가지 수준의 단위를 사용한다.

276 This family uses two levels of decomposition: the *subsystem* and the *module*. A module is the most specific description of functionality: it is a description of the implementation. A developer should be able to implement the part of the TOE described by the module with no further design decisions. A subsystem is a description of the design of the TOE; it helps to provide a high-level description of what a portion of the TOE is doing and how. ...

277 The general approach adopted for design documentation is that, as the level of assurance increases, the emphasis of description shifts from the general (subsystem level) to more (module level) detail. ...

(그림 42) 서브시스템과 모듈 정의에 대한 CC 원문

서브시스템은 ADV_TDS에 대한 구조적인 설명으로 TOE를 구성하는 서브시스템이 어떤 기능을 수행하는지 상위 수준으로 설명하고, 모듈은 기능을 어떻게 수행하는지 하위 수준에서 상세히 기술한다. ADV_TDS에서 가장 상세한 수준의 기술이 모듈로서 더 이상의 설계 과정 없이 구현될 수 있는 단위이며 원자성(atomic) 특징을 가진 기능성을 포함하는 단위이다.

소프트웨어 분석하는 과정 또는 초기(initial) 아키텍처를 설계 하는 과정에서 추출되는 기능성(functionality)은 비교적 단위(coarse-grained)가 크며, CC에서의 서브시스템 단위가 대

부분 도출될 것이다. 이는 소프트웨어 개략 설계/상세 설계 과정을 거치거나 초기 아키텍처를 정제하는 과정을 거치면서, 기능성이 세분화되고 기능성 단위가 작아지며(fine-grained), 최종적으로는 한 기능 단위가 원자성 기능을 실체화하게 된다.

소프트웨어 공학에서 아키텍처 명세서는 뷰에 따라 전제 시스템의 구조를 구성요소(컴포넌트와 컴포넌트 간 관계)를 아키텍처 스타일을 이용해서 기술하고, 설계 근거도 상세히 기술한다. 이 명세서에는 아키텍처 분석 또는 설계 수준에 따라 서브시스템 또는 모듈이 명세 구성요소가 될 수 있다. 예를 들어, 분석 수준의 아키텍처 명세서는 기능성 단위가 큰 컴포넌트들이 주로 추출되기 때문에 서브시스템을 단위로 기술할 수 있고, 설계 수준의 아키텍처 명세서는 컴포넌트가 상세화되어 클래스 단위의 모듈 단위로 기술할 수 있다.

아키텍처 명세서 뿐 아니라 소프트웨어 분석 또는 설계서에도 객체, 클래스, 컴포넌트, 서브시스템 등의 단위로 소프트웨어를 기술 할 수 있는데 이들도 역시 상세도에 따라 서브시스템과 모듈을 통칭하는 용어가 될 수 있다. 서브시스템과 모듈은 보증 수준에 따라 서술의 초점이 서브시스템 수준에서 모듈 수준으로 이동한다. 이는 설계서의 내용 상세도 수준과 연관되어 있는 것으로, 보증 수준이 높다라는 의미는 해당 보증 요구사항을 소프트웨어가 높은 수준으로 구현하고 있다는 것을 의미하며 이는 곧 보증 요구사항에 대한 설계 수준이 상세하다라는 것을 의미한다.

소프트웨어 공학에서는 “추상화(abstraction)”이라는 개념이 실세계의 복잡한 문제를 정의하는데 가장 기본이 되는 방법으로, 가장 높은 추상화 단계에서 복잡한 문제를 개략적으로 설명하고 추상화 단계가 낮아지면서 점점 상세하기 문제를 기술한다. 서브시스템과 모듈 단계는 소프트웨어 공학에서의 추상화 기법과 유사하다. 즉, 상위 수준에서 TOE 설계가 어떻게 구성되는지를 서브시스템 단위로 기술하고 이를 더 상세하기 기술하기 위해 모듈 단위를 사용한다.

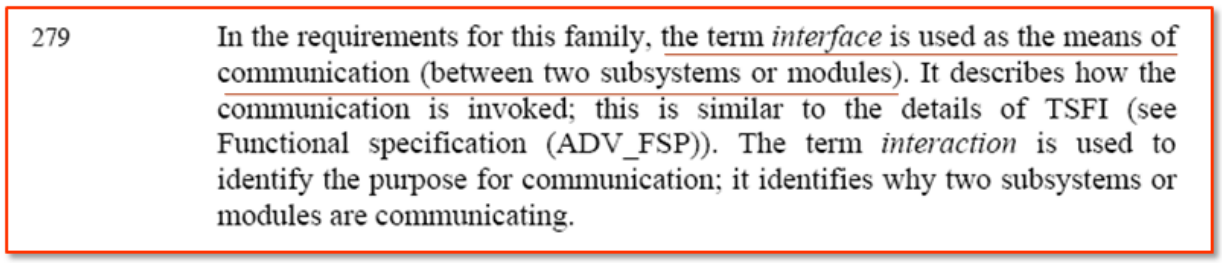
소프트웨어 요구사항은 물론 분석/설계서는 소프트웨어를 테스팅할 때 유용한 자료가 된다. (그림 42)에서 언급된 것와 같이, SFR이 정확하게 구현되어 있다는 것을 입증하기 위해서는 상세한 수준의 설계서가 필요하며 이는 해당 소프트웨어를 테스트할 때 사용될 수 있다.

278	This approach follows the general paradigm that providing additional detail about the implementation of the TSF will result in greater assurance that the SFRs are implemented correctly, and provide information that can be used to demonstrate this in testing (ATE: Tests).

(그림 43) 설계서 상세도과 보증 수준에 대해 설명하는 CC 원문

소프트웨어의 컴포넌트 또는 객체가 각자 기능을 수행하기 위해서는 다른 컴포넌트 또는 객체와 상호작용을 해야 하는 경우가 있다. 이 개념은 CC에서도 적용되어, 서브시스템 또는 모듈은 각자 기능을 수행하기 위해 다른 서브시스템 또는 모듈과 상호작용을 해야 한다.

이런 경우 인터페이스가 상호작용하기 위한 수단으로 사용된다 (그림 44).


279 In the requirements for this family, the term *interface* is used as the means of communication (between two subsystems or modules). It describes how the communication is invoked; this is similar to the details of TSFI (see Functional specification (ADV_FSP)). The term *interaction* is used to identify the purpose for communication; it identifies why two subsystems or modules are communicating.

(그림 44) 인터페이스와 상호작용을 설명하는 CC 원문

인터페이스의 중요성은 절차적인 방법론에서 객체지향 방법론(Object Oriented Analysis and Design, OOA/D)으로 진화하면서 대두되기 시작하였으며, 컴포넌트 기반 개발 방법론(Component-based Development, CBD)에서는 인터페이스가 컴포넌트 구현과 완전히 분리되어 정의된다. OOA/D에서는 private한 정보는 내부에 숨겨놓고 public 인터페이스를 통해서만 private 정보를 접근 가능하도록 하였지만 한 클래스에 private 데이터와 public 인터페이스를 정의하였다. 그러므로, OO에서는 public 오퍼레이션 시그니처로 클래스가 서로 상호작용을 한다.

CBD 개념이 생기면서 컴포넌트와 인터페이스의 기능이 더 강조되고 완전하게 분리하여 인터페이스를 통해서만 컴포넌트의 기능을 접근할 수 있도록 하였다. CBD에서는 인터페이스는 컴포넌트가 제공하는 기능을 명세하는 Provided 인터페이스와 컴포넌트가 필요로 하는 기능을 명세하는 Required 인터페이스로 분류된다.

소프트웨어를 분석/설계/구현하는 동안에는 관련된 산출물을 만든다. 이 산출물은 향후 소프트웨어 테스팅 등에 유용하게 쓰인다. 보증 요구사항을 평가하기 위해 해당 요구사항이 어떻게 설계/구현 되어 있는지를 확인해야 하며, 이를 위해 그림 44와 같이 서브시스템과 모듈을 명세하는 항목을 기술하였다. CC에서는 서브시스템의 식별 가능한 고유한 이름과 종류(SFR-수행, SFR-지원, SFR-비-간섭), 서브시스템 행위에 대한 설명, 다른 서브시스템과 어떻게 상호작용하는지를 설명하는 항목, 모듈의 상세 설명으로 구성된다.

소프트웨어 공학에서는 산출물 종류에 따라 CC에서 언급한 명세 항목보다 더 상세하게 요소(컴포넌트, 클래스 등)을 기술한다.

280 The requirements define collections of details about subsystems and modules to be provided:

- The subsystems and modules are *identified* with a simple list of what they are.
- A subsystem may be *categorised* (either implicitly or explicitly) as
- A subsystem's *behaviour* is what it does.
- A *behaviour summary* of a subsystem is ...
- A *behaviour description* of a subsystem is ...
- A *description of interactions* among or between subsystems ...
- The *purpose* of a module provides sufficient detail ...

(그림 45) 서브시스템과 모듈의 명세 항목에 대해 설명하는 CC 원문

그림 44에서 도출한 항목들은 서브시스템 또는 모듈에 상관없이 동일하게 기술되는 명세 항목이지만, 서브시스템 또는 모듈에 따라 기술되는 상세도 정도가 달라질 수 있다. 이는 소프트웨어 공학에서도 동일하게 적용된다. 예를 들어, 분석 수준의 아키텍처 명세서와 설계 수준의 아키텍처 명세서를 비교하면, 두 명세서에 기술되는 요소(컴포넌트 단위 또는 클래스 단위)의 기능성 단위가 달라짐에 따라 이에 대한 설명 상세도 수준이 달라진다.

이번 절에서는 서브시스템과 모듈 공통적으로 적용되는 사항 외의 각각의 특성을 소프트웨어 공학적으로 분석한다.

1. 서브시스템 (Subsystem)

서브시스템은 ADV_TDS에 대한 구조적인 설명으로 TOE를 구성하는 서브시스템이 어떤 기능을 수행하는지 상위 수준에서 기술한다. 서브시스템은 더 상세 수준의 서브시스템 또는 모듈로 세분화할 수 있으며, 그림 46과 같이 TSF의 경계를 분명하게 하고 시스템의 구조와 각 요소의 기능을 보여주는데 초점이 맞추어져 있다.

578 The first use of subsystems is to distinguish the TSF boundary; that is, the portions of the TOE that comprise the TSF. In general, a subsystem is part of the TSF if it has the capability (whether by design or implementation) to affect the correct operation of any of the SFRs. ...

579 The second use of subsystems is to provide a structure for describing the TSF at a level of description that, while describing how the TSF works, does not necessarily contain low-level implementation detail found in module ...

(그림 46) 서브시스템의 목적을 설명하는 CC 원문

소프트웨어 생명주기 별로 생성되는 산출물의 기능성 단위를 고려하면, 분석 단계의 산출물의 기능이 대부분 서브시스템 단위로 도출되며 분석 단계의 목적은 사용자가 제공하는 요구사항의 기능을 분석하고 개발하고자 하는 소프트웨어(CC에서는 TOE)의 경계를 분명하게 하는 것이다. 이런 관점에서 CC에서 언급하는 서브시스템의 목적은 소프트웨어 분석 단계에서 나오는 산출물의 기능성 단위 수준과 일치한다고 볼 수 있다.

즉, 서브시스템 단위로 ADV_TDS를 명세하는 것은 소프트웨어 개발 프로세스에서 소프트웨어 요구사항으로부터 시스템의 기능을 분석하는 단계의 산출물과 비슷하다. 그러므로, 분석 단계에서 만든 산출물은 서브시스템 단위로 ADV_TDS를 명세하는 것에 적용될 수 있다. 다만 분석 산출물과의 가장 큰 차이점은 서브시스템의 서술 수준은 해당 서브시스템이 SFR 구현할 책임 정도(SFR-수행, SFR-지원, SFR-비-간섭)에 따라 결정되는 반면에 분석 산출물은 한 소프트웨어에 관한 분석 산출물은 기능성 특성 또는 분류 기준에 상관없이 비슷한 수준으로 서술된다는 것이다.

그림 47은 서브시스템에 대한 명세 항목 중 일부인 서브시스템의 분류(종류) 항목에 대한 설명이다. TOE를 설계하기 위해서는 요구사항에 필요하며, 요구사항은 SFR을 지원하는지에 따라 SFR-수행(SFR-enforcing), SFR-지원(SFR-supporting), SFR-비-간섭(SFR-non-interfering)으로 분류되고 이 분류 기준은 서브시스템에도 동일하게 적용된다. 그러므로, 그림 47와 같이 기능별로 분류된 서브시스템은 2차적으로 보안기능과 관련된 요구사항을 설계한 것인지에 따라 SFR-수행, SFR-지원, SFR-비-간섭 서브시스템으로 분류된다. CC에서는 서브시스템의 종류에 따라 기술 상세도 수준이 다르므로, 이 정보를 중요하게 다루며 이 정보는 명세서에 꼭 기입되어야 한다.

- A subsystem may be *categorised* (either implicitly or explicitly) as "SFR-enforcing", "SFR-supporting", or "SFR-non-interfering"; these terms are used the same as they are used in Functional specification (ADV_FSP).

(그림 47) 서브시스템 분류에 관한 CC 원문

소프트웨어 요구사항은 기능성 요구사항(Functional Requirement) 뿐 아니라 비기능성 요구사항(Non-functional Requirement)도 포함한다. 그리고 비기능성 요구사항은 보안성, 신뢰성 등과 같은 소프트웨어의 품질 속성과 관련된다. 소프트웨어 공학에서는 소프트웨어를 기능별로 분류할 때 결합도(coupling)와 응집도(cohesion) 기준을 사용한다. CC에서도 응집도가 높고 다른 서브시스템과의 결합도가 낮도록 서브시스템이 1차적으로 식별되며, 보안 기능사항(SFR)에 따라 2차적으로 분류되는 것이다. CC는 보안성은 평가하기 위한 아키텍처와 관련 있으므로, 기능성 요구사항 뿐이니라 비기능성 요구사항을 중요하게 고려하며 이를 기준으로 SFR-수행, SFR-지원, SFR-비-간섭으로 분류하는 것이다.

- SFR-수행 : SFR 기능을 수행하는데 직접적인 역할을 하는 서브시스템
- SFR-지원 : SFR 기능을 수행하는데 도움은 주지만 직접적인 역할은 하지 않은 서브시스템
- SFR-비-간섭 : SFR 기능을 수행하는데 직접적인 역할 또는 지원하는 역할로써 의존되지 않은 서브시스템

그림 48와 같이 서브시스템의 행위는 서브시스템이 어떤 기능을 행하는지를 기술한 것으로, 서브시스템 분류와 마찬가지로 SFR-수행, SFR-지원, SFR-비-간섭으로 분류된다. 각 서브시스템의 행위는 SFR을 수행하기 위해 행해지는 활동이기 때문에 SFR 분류와 밀접한 연관이 있다. 그러나, SFR-수행 서브시스템일지라도 모든 행위가 SFR 기능을 수행하는 행위만 존재하는 것이 아니기 때문에 서브시스템의 분류와 서브시스템의 행위 분류는 1:1로 매핑이 되는 것이 아니다. 즉, SFR-수행 서브시스템일지라도 그 행위 중 일부는 SFR-수행 행위가 아닐 수도 있다.

- A subsystem's *behaviour* is what it does. The behaviour may also be categorised as SFR-enforcing, SFR-supporting, or SFR-non-interfering. The behaviour of the subsystem is never categorised as more SFR-relevant than the category of the subsystem itself. For example, an SFR-enforcing subsystem can have SFR-enforcing behaviour as well as non-SFR-enforcing behaviour.

(그림 48) 서브시스템 행위에 관한 CC 원문

그림 49과 같이 서브시스템 간의 상호작용 명세는 두 서브시스템이 서로 상호작용하여 기능을 수행하기 위해 필요한 정보를 포함한다. 그러므로, 이 명세에는 서브시스템이 서로 상호작용하는 이유를 식별하고, 상호작용 할 때 전송되는 데이터를 기술한다.

서브시스템 단위에서의 기술은 기능을 어떻게 제공해주는지에 초점이 맞추어지지 않기 때문에, 상세한 수준의 기술은 필요로 하지 않다. 그러므로, 두 서브시스템이 주고받는 데이터의 매개변수와 매개변수의 데이터 타입, 반환 값과 반환값의 데이터 타입 등의 상세한 정보 명세는 요구되지 않는다.

- A *description of interactions* among or between subsystems identifies the reason that subsystems communicate, and characterises the information that is passed. It need not define the information to the same level of detail as an interface specification. For example, it would be sufficient to say "subsystem X requests a block of memory from the memory manager, which responds with the location of the allocated memory."

(그림 49) 상호작용 명세에 대한 CC 원문

Use Case 명세서에서 여러 모듈들이 상호작용하더라도 상세히 어떻게 상호작용하는지 기술하지 않고 상호작용하는데 사용되는 데이터 정보만 기술하기 때문에, 서브시스템 간의 상호작용 명세는 Use Case 명세서에서 기능을 수행하기 위한 흐름을 작성하는 수준과 비슷하다.

1절 설명에 앞서 언급하였듯이, 서브시스템에 대한 명세 항목은 서브시스템의 식별 가능한 고유한 이름과 종류(SFR-수행, SFR-지원, SFR-비-간섭), 서브시스템 행위에 대한 설명, 다른 서브시스템과 어떻게 상호작용하는지를 설명하는 항목, 모듈의 상세 설명으로 구성된다. 소프트웨어 아키텍처는 대부분 하나 이상의 여러 뷰를 통하여 명세된다. 대부분 아키텍처는 기능적인(Functional) 뷰, 정보(Information) 뷰, 동시(Concurrency) 뷰, 개발(Development) 뷰, 배치(Deployment) 뷰, 운영(Operatioanl) 뷰를 이용하여 명세된다.

기능적인 뷰에는 시스템의 기능적인 요소와 이 요소들의 기능성, 인터페이스, 주요 상호작용 정보를 포함하며, CC에서 언급한 명세 항목 중 서브시스템의 이름과 종류, 서브시스템간 상호작용 설명을 기술할 수 있을 것이다. 정보 뷰에는 소프트웨어가 관리하는 정보 또는 데이터를 나타내며, CC 항목 중에는 이 뷰에 포함될 정보는 없다. 동시 뷰는 소프트웨어가 실시간에 어떻게 상호작용하는지를 보여주며, CC의 서브시스템 간 상호작용 정보를 기능적인 뷰와 다른 관점에서 기술한다. 개발 뷰는 소프트웨어 개발 프로세스를 지원하는 정보를 기술하고, 배치 뷰는 소프트웨어가 배치될 환경 정보를 기술하며, 운영 뷰는 시스템이 어떻게 운영되는지를 기술하지만 CC에는 이 뷰에 기술될 명세항목은 언급되어 있지 않다.

ADV_TDS의 수준(1~6)에 따라 서브시스템을 기술하는 방법은 다르지만, 텍스트 형태 외에 다양한 다이어그램을 이용하여 소프트웨어 아키텍처를 표현할 수 있다. 아키텍처 뷰 별로 사용되는 다이어그램은 각 시스템마다 차이가 있을 수 있다. 하지만, 기능적인 뷰에서 소프트웨어의 구조를 보여주기 위해 사용되는 패키지 다이어그램 또는 컴포넌트 다이어그램이 사용될 수 있다. 그리고, 정보 뷰를 표현하기 위해 클래스 다이어그램 또는 데이터 흐름 다이어그램(Data Flow Diagram, DFD), 동시 뷰를 위해서는 컴포넌트 다이어그램의 메시지 호출 관계 표현 또는 시퀀스 다이어그램이 사용될 수 있다. 그리고, 배치 뷰를 표현하기 위해서는 배치 다이어그램 등이 사용될 수 있다.

2. 모듈 (Module)

모듈은 기능성을 가장 상세한 수준에서 기술하는 것으로, 더 이상의 설계 과정이 필요하지 않을 정도의 세부 사항을 충분히 제공하는 것이다. 모듈의 목적은 그림 50과 같이 해당 모듈이 어떤 기능을 어떻게 제공하는지에 기술하는 것이다.

588	The purpose of a module should be described indicating what function the module is providing. It should be sufficient so that the reader could get a general idea of what the module's function is in the architecture.

(그림 50) 모듈 목적을 설명하는 CC 원문

서브시스템의 분류 기준과 마찬가지고, SFR-수행(SFR-enforcing), SFR-지원(SFR-supporting), SFR-비-간섭(SFR-non-interfering)으로 분류되고 이 분류 기준은 모듈에도 적용된다. 그러므로, 그림 51과 같이 기능별로 분류된 모듈은 2차적으로 보안기능과 관련된 요구사항을 설계한 것인지에 따라 SFR-수행, SFR-지원, SFR-비-간섭 서브시스템으로 분류된다. 서브시스템 보다는 더 상세한 수준의 단위이지만, 모듈 역시 결합도와 응집도 기준을 사용하여 식별된다. 그러므로 이미 식별된 서브시스템 중에서, 응집도가 높고 다른 모듈과의 결합도가 낮은 모듈이 좀 더 세분화되어 식별되며, 보안 기능사항 (SFR)에 따라 2차적으로 분류되는 것이다.

584	An SFR-enforcing module is a module that directly implements a security functional requirement (SFR) in the ST. Such modules will typically implement an SFR-enforcing TSFI, but some functionality expressed in an SFR (for example, audit and object re-use functionality) may not be directly tied to a single TSFI. ...

(그림 51) 모듈 분류에 대한 CC 원문

- o SFR-수행 : SFR 기능을 수행하는데 직접적인 역할을 하는 모듈
- o SFR-지원 : SFR 기능을 수행하는데 도움은 주지만 직접적인 역할은 하지 않은 모듈
- o SFR-비-간섭 : SFR 기능을 수행하는데 직접적인 역할 또는 지원하는 역할로써 의존되지 않은 모듈

모듈 단위로 ADV_TDS를 명세하는 것은 소프트웨어 개발 프로세스에서 소프트웨어 요구사항으로부터 시스템의 기능을 설계하는 단계의 산출물과 비슷하여 구현에 대해 기술한다.

그러므로, 설계 단계에서 만든 산출물은 모듈 단위로 ADV_TDS를 명세하는 것에 적용될 수 있다. 그림 52는 모듈 명세 항목을 설명한 부분이다.

587	In the design, modules are described in detail in terms of the function they provide (the purpose); the interfaces they present; the return values from such interfaces; the interfaces (presented by other modules) they use; and an algorithmic description of how they provide their functionality.

(그림 52) 모듈 명세 항목에 대한 CC 원문

모듈 명세 항목은 모듈이 제공하는 기능(모듈의 목적), 기능에 대한 인터페이스(Interface they present), 모듈이 사용하는 다른 모듈의 인터페이스(Interface they use), 반환값, 기능을 어떻게 제공하는지에 대한 알고리즘이 있다. 앞서 마찬가지고 이는 설계서의 명세 항목과 비슷하므로 OO나 CBD 방법론에 따라서 기술되는 항목이 약간 다르다. ADV_TDS에서 기술된 인터페이스의 종류는 CBD의 인터페이스와 비슷하게 분류된다. 즉, 기능에 대한 인터페이스(Interface they present)는 CBD에서 컴포넌트가 제공하는 기능을 명세하는 Provided 인터페이스와 비슷하고, 모듈이 사용하는 다른 모듈의 인터페이스(Interface they use)는 CBD에서 컴포넌트가 필요로 하는 기능을 명세하는 Required 인터페이스와 비슷하다.

OO에서는 클래스 단위로 설계 명세서를 작성하며, 다음과 같은 항목과 같이 소프트웨어의 동적인 구조와 정적인 구조를 분류하여 설계서를 작성된다.

- o 클래스 다이어그램 : 클래스의 정적 구조를 보여줌.
  - 각 클래스의 이름 및 개략적인 기능 설명
  - 클래스의 속성 설명 : 속성 데이터 타입과 범위값 설명
  - 클래스의 오퍼레이션 설명 : 오퍼레이션 시그니처(오퍼레이션 이름, 반환값, 매개변수 정보)와 기능 설명 - public 오퍼레이션은 인터페이스 역할을 함.
- o 시퀀스 다이어그램 : 클래스 간의 상호작용(동적 구조)을 보여줌 .

CBD에서는 컴포넌트와 인터페이스를 분리해서 작성하며, 컴포넌트 명세서는 OO에서와 같이 소프트웨어의 동적인 구조와 정적인 구조를 모두 반영하도록 다음과 같은 항목으로 기술된다.

- o 컴포넌트 이름
- o 컴포넌트 설명
  - 컴포넌트의 개요 및 기능 기술 (기능적 설명과 비기능적 설명)
  - 컴포넌트를 수행하기 위해 필요한 입력 데이터와 기능 수행 후의 결과값

o 컴포넌트 인터페이스
  - 해당 컴포넌트의 기능을 기술한 인터페이스 이름
  - 컴포넌트가 사용하는 다른 컴포넌트의 인터페이스 이름
o 컴포넌트의 구조적 모델링
  - 컴포넌트를 구성하는 클래스 다이어그램
  - 클래스 별 기능 설명
  - 클래스를 구성하는 속성과 오퍼레이션 기술
o 컴포넌트의 동적 모델링
  - 클래스들 간의 상호작용을 보여주는 시퀀스(Sequence) 다이어그램이나 커뮤니케이션(Communication) 다이어그램을 이용

컴포넌트 명세서 외에 CBD에서는 인터페이스 명세서도 별도로 기술하며, 다음과 같은 항목으로 구성된다.

o 오퍼레이션 시그니처
  - 오퍼레이션 이름, 매개변수(이름, 데이터 타입), 매개변수 값의 범위, 리턴 타입, 리턴 타입의 값의 범위 등
o 오퍼레이션이 제공하는 기능성에 관해 기술
o 오퍼레이션의 시맨틱 정보 기술
  - 선행 조건(Precondition), 후행 조건(Postcondition), 불변치(invariant), 제약 조건(constraint)
  - 텍스트 또는 OCL을 이용하여 기술 가능

ADV_TDS에서는 위와 같이 방법론에 따라 상세한 기술 항목들을 나열하고 있지는 않지만, 위의 항목들은 ADV_TDS를 기술하는데 적용될 수 있으며 ADV_TDS.1부터 ADV_TDS.6까지 각 명세 항목은 다른 상세도 수준과 표현 기법을 이용하여 기술한다.

서브시스템과 마찬가지로, 그림 53과 같이 모듈도 다른 모듈과 상호작용하기 위해 인터페이스를 이용하고 이 역시 ADV_TDS에 기술되어야 한다. ADV_TDS의 모듈 수준의 인터페이스는 호출되는 방법, 반환값, 매개변수에 대한 정보를 기술한다.

589 The interfaces presented by a module are those interfaces used by other modules to invoke the functionality provided. Interfaces include both *explicit* interfaces (e.g., a calling sequence invoked by other modules) as well as *implicit* interfaces (e.g., global data manipulated by the module). Interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. If a parameter were expected to take on a set of values (e.g., a "flag" parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are

(그림 53) 모듈 수준의 인터페이스 명세에 대한 CC 원문

앞서 말했듯이, 모듈 수준의 ADV_TDS는 소프트웨어 설계와 많이 비슷하며 인터페이스 정보도 명시적 또는 암시적으로 기술된다. OO에서는 오퍼레이션 시그니처로 두 개의 클래스가 상호작용을 하며, 오퍼레이션 이름, 매개변수(타입, 이름), 반환값, 반환값 데이터 타입 등의 정보가 기술된다. CBD에서는 인터페이스의 종류가 분류(Provided 인터페이스, Required 인터페이스)되며, 각 인터페이스 종류에 따라 되어 각각 다르게 기술되지만 공통적으로는 오퍼레이션 시그니처, 오퍼레이션의 기능 설명, 시맨틱 정보(사후 조건, 사전 조건, 불변치, 제약 조건 등)이 명세 항목에 포함된다.

모듈 수준의 ADV_TDS는 설계 과정이 더 이상 필요하지 않을 정도로 세부사항을 기술하기 때문에, 그림 54과 같이 입력 데이터를 이용하여 기능성이 어떻게 기술되는지에 대한 정보(알고리즘)도 상세한 수준으로 기술되어야 한다.

ADV_TDS에서 알고리즘 명세는 ADV_TDS 계층 (1~6)에 따라 Pseduo-code, Flow Chart, 텍스트 형식 이용하여 기술하며, 입력 데이터를 기반으로 모듈의 기능을 어떻게 수행하는지 상세하게 기술된다.

592 As discussed previously, the algorithmic description of the module should describe in an algorithmic fashion the implementation of the module. This can be done in pseudo-code, through flow charts, or (at ADV_TDS.3 Basic modular design) informal text. It discusses how the module inputs and called functions are used to accomplish the module's function. It notes changes to global data, system state, and return values produced by the module. It is at the level of detail that an implementation could be derived that would be very similar to the actual implementation of the TOE.

(그림 54) 모듈의 알고리즘 명세에 대한 CC 원문

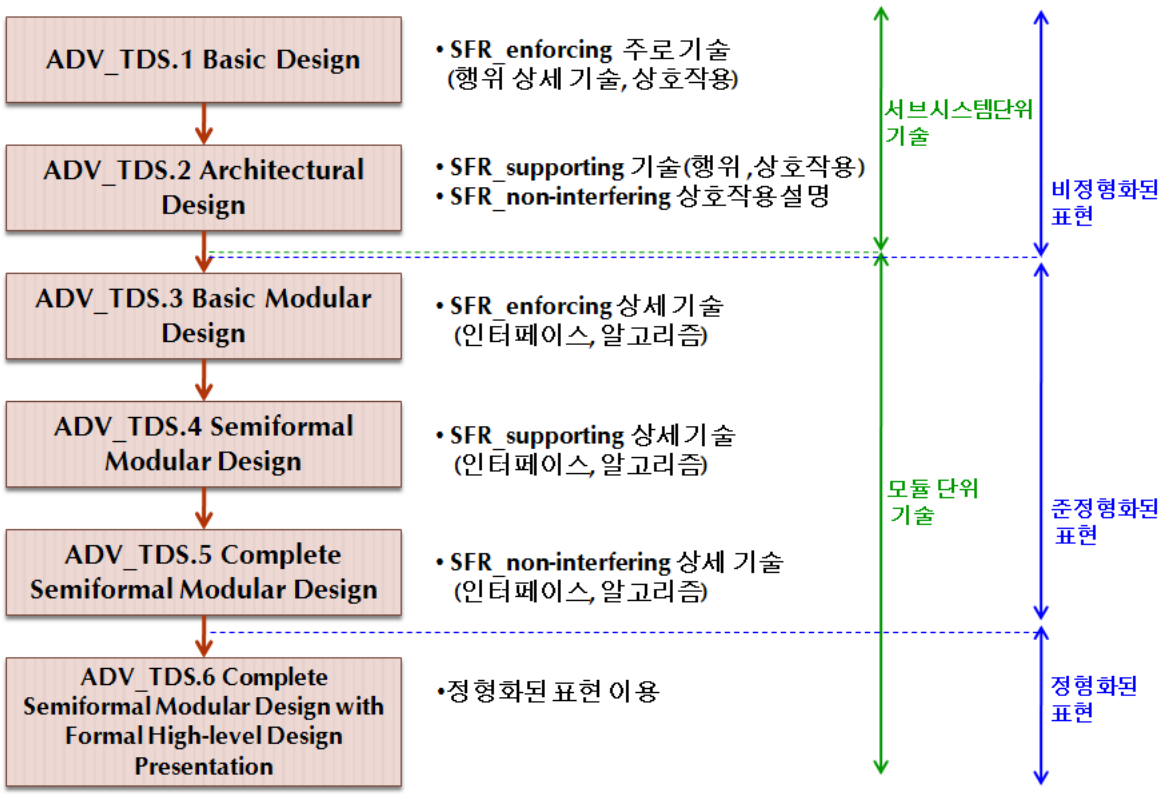
소프트웨어 설계서에도 이와 비슷한 방식으로 알고리즘을 명세할 수 있으며, Pseduo-code, Flow Chart, 텍스트 형식 외에 UML 표기법을 이용하기도 한다. 알고리즘을

명세하기 위해서는 상세한 수준은 시퀀스 다이어그램이나 커뮤니케이션 다이어그램 등이 사용될 수 있다.

이렇듯 전반적으로 사용되는 다이어그램은 서브시스템을 기술하기 위해 사용된 다이어그램과 비슷하지만, 상세도 수준이 다르다. 예를 들어, 기능적인 뷰에서 모듈 수준의 설계서를 기술할 때는 컴포넌트 다이어그램 보다는 클래스 다이어그램으로 표현할 수 있으며 상세하게 클래스의 속성과 오퍼레이션 정보도 상세하게 명세한다.

제 4 절 컴포넌트 계층관계

보증 등급이 높아짐에 따라 더 상세한 정보를 요구할 수 있기 때문에 따라 ADV_TDS 패밀리는 그림 55와 같이 6단계의 컴포넌트로 분류된다.



(그림 55) ADV_TDS의 계층화

ADV_TDS에서는 2가지 기준을 이용하여 6개의 컴포넌트를 분류하였는데, 첫 번째 기준은 기술되는 단위이고 두 번째 기준은 표현 기법이다. 첫 번째 기준에 의하여 ADV_TDS.1와 ADV_TDS.2는 서브시스템 단위로 개략적으로 TOE에 대해서 기술하지만, ADV_TDS.3부터 ADV_TDS.6까지는 모듈 단위로 TOE를 상세하게 기술한다. 그리고 두 번째 기준에 의하여 ADV_TDS.1부터 ADV_TDS.3까지는 비정형화된 표현을 이용하여 TOE 설계를 기술하는

반면에, ADV_TDS.4와 ADV_TDS.5는 준정형화된 표현, ADV_TDS.6은 준정형화된 표현과 정형화된 표현을 이용해서 TOE 설계를 명세한다. 이리하여 ADV_TDS.1에서 ADV_TDS.6로 계층이 높아질수록 TOE 설계는 좀 더 상세한 내용을 더 정확하게 표현할 수 있게 된다.

ADV_TDS 패밀리의 6개의 모든 컴포넌트에 공통으로 기술된 엘리먼트가 있다. 그리고, 컴포넌트의 계층이 높아질수록 전 단계에서 정의된 엘리먼트가 그대로 적용되거나 재정의 된다.

그림 56은 개발자 요구사항을 보여준다. CC에서는 TOE 설계를 제공하는 사람이 개발자(developer)로 정의되어 있지만, 소프트웨어 공학에서는 분석/설계/구현하는 사람의 역할이 분리되어 있어 각각 분석가(Analyst), 설계자(Designer), 프로그래머(Programmer), 아키텍처 설계자(Architect)로 불린다.

ADV_TDS.1.1D	**The developer shall provide the design of the TOE.**
ADV_TDS.1.2D	**The developer shall provide a mapping from the TSFI of the functional specification to the lowest level of decomposition available in the TOE design.**

(그림 56) 개발자 요구사항에 대한 CC 원문

이 분리된 역할을 CC에 적용하게 되면, ADV_TDS의 계층 컴포넌트마다 약간의 차이가 존재할 수 있지만 소프트웨어 분석가가 서브시스템 단위의 명세서를 작성하게 될 것이고 설계자가 모듈 단위의 명세서를 작성하게 된다 (ADV_TDS.3 단계 이상). 그리고 아키텍처 설계자는 소프트웨어 분석 또는 설계 단계에 따라 서브시스템 단위와 모듈 단위로 이루어진 아키텍처 명세서를 작성한다.

그림 57와 같이 ADV_TDS 패밀리의 모든 컴포넌트에 평가자 요구사항도 같은 내용으로 정의되어 있다.

평가자는 TOE 설계의 모든 정보들이 모든 증거 요구사항(content and presenatation fo evidence)을 만족하는지를 확인해야 하고, 모든 보안 요구사항 (SFR)이 정확하고 완전하게 설계에 포함되어 있는지를 결정해야 한다.

ADV_TDS.1.1E	The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.
ADV_TDS.1.2E	The evaluator *shall determine* that the design is an accurate and complete instantiation of all security functional requirements.

(그림 57) 평가자 요구사항에 대한 CC 원문

앞서 1절 설계 명세서의 충분성에서 언급하였듯이, 소프트웨어 공학에서는 정확성과 완전성 역시 평가하기 힘든 항목 중 하나이다. 완전성을 평가하기 위해서는 완전성에 대한 기준을 먼저 제시해야 하는데, 이 역시 매우 어려운 작업이다.

소프트웨어 공학에서는 추적성(Traceability)을 이용하여 소스코드 상에 있는 모든 구현이 요구사항을 만족하는지 평가할 수 있다. 이는 소프트웨어 개발 프로세스의 각 단계를 거치면서 요구사항 명세서에 있는 모든 요구사항들이 구현에 반영되어 있는지 확인하는 작업으로 산출물간의 추적이 가능해지면 소프트웨어의 일관성을 유지하는 것이 가능해진다.

소프트웨어 공학에서 이와 비슷한 목적의 품질 보증 활동으로 확인(Validation)과 검증(Verification)이 있다. 확인은 올바른 소프트웨어를 만들고 있는지 확인하는 작업으로 요구사항을 충족시키도록 소프트웨어를 개발하는지를 평가한다. 검증은 소프트웨어를 올바르게 만들고 있는지 평가하는 작업으로, 소프트웨어 개발 주기에서 주어진 단계에서의 생산품이 이전 단계에서 수립된 요구들을 수행하는가를 평가한다.

다음부터 ADV_TDS 패밀리의 각 컴포넌트의 특성을 알아보고 이를 소프트웨어 공학적으로 분석한다.

1. ADV_TDS.1 기본적인 설계 (Basic Design)

ADV_TDS.1 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 서브시스템 단위로 기술함.
- o SFR-수행 서브시스템은 중점으로 기술함.
  - 아키텍처, 상위 수준으로 작성된 SFR- 수행 행위, 상호작용
- o SFR-지원과 SFR-비-간섭 서브시스템은 서브시스템의 분류에 대한 근거를 제시할 수 있는 정도로만 기술함.
- o 모든 내용은 비정형화된 표현으로 기술됨.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.1 Basic design (informal presentation)	architecture, high-level description of SFR-Enf. behaviour, interactions	designation support[1]	designation support			

(그림 58) ADV_TDS.1 에 기술되는 정보

이 단계 컴포넌트에서는 기능 설명, 행위 설명, 상호작용 설명이 모두 비정형화된 표현 기법으로 이루어지기 때문에, 개발자가 SFR 분류 근거, 각 SFR을 어떻게 설계하였는지, 설계 근거에 대해서 자연어(텍스트)로 논리적으로 기술해야 한다.

이 단계에서는 SFR을 직접적으로 수행하는 서브시스템에 대해서만 아키텍처, 상위 수준에서의 행위와 상호작용 명세서를 텍스트 형식으로 기술하고, 그 외 다른 서브시스템에 대해서는 SFR-수행 서브시스템을 어떻게 지원하는지에 대해서만 기술한다.

2. ADV_TDS.2 구조적인 설계 (Architectural Design)

ADV_TDS.2 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 서브시스템 단위로 기술함.
- o 서브시스템들이 어떻게 상호작용 하는지 기술함.
- o SFR-수행 서브시스템은 다음과 같이 기술함.
  - 구조, 상세하게 작성된 모든 행위
- o ADV_TDS.1과 달리 SFR-지원 서브시스템에 대해서도 간략히 기술함.
  - 구조, 상위 수준으로 기술된 행위
- o SFR-비-간섭 서브시스템은 서브시스템의 분류에 대한 근거를 제시할 수 있는 정도로만 기술함.
- o 모든 내용은 비정형화된 표현으로 기술됨.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.2 Architectural design (informal presentation)	architecture, detailed description of SFR-Enf. behaviour, high-level description of other behaviour interactions	architecture, high-level description of behaviour, interactions	designation support, interactions			

(그림 59) ADV_TDS.2 에 기술되는 정보

ADV_TDS.1과 마찬가지로 ADV_TDS.2 컴포넌트에서는 기능 설명, 행위 설명, 상호작용 설명이 모두 비정형화된 표현 기법으로 이루어지며, 개발자가 SFR 분류 근거, 각 SFR을 어떻게 설계하였는지, 설계 근거에 대해서 자연어로 논리적으로 기술해야 한다. ADV_TDS.1과 달리 SFR-지원 서브시스템에 대한 정보도 충분히 기술한다는 것만 다르지, 소프트웨어 공학적으로는 이 단계의 설계 명세 기법은 ADV_TDS.1 과 비슷할 것이다.

3. ADV_TDS.3 기본적인 모듈화 설계 (Basic Modular Design)

ADV_TDS.3 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 모듈 단위로 기술함.
- o 서브시스템 설명 기술함.
- o 모듈들이 어떻게 상호작용 하는지 기술함.
- o SFR-수행 모듈은 상세하게 기술함.
  - 기능 설명, 데이터, 인터페이스, 알고리즘 포함
- o SFR-지원과 SFR-비-간섭 모듈은 기능 설명만 간략히 기술함.
- o 모든 내용은 준정형화된 표현으로 기술됨.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.3 Basic modular design (informal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces(2), algorithmic(3)	interaction, purpose	interaction, purpose

(그림 60) ADV_TDS.3 에 기술되는 정보

전 단계의 컴포넌트와 마찬가지로, ADV_TDS.3 컴포넌트로 비정형화된 방법으로 기술되며 개발자가 SFR 분류 근거, 각 SFR을 어떻게 설계하였는지, 설계 근거에 대해서 자연어로 논리적으로 기술해야 한다. 이 단계의 컴포넌트는 이전 단계와는 달리 모듈 단위로 기술되기 때문에, 데이터, 인터페이스 정보, 알고리즘 등 SFR-수행 모듈을 더 상세하게 기술한다. 그렇지만, 비정형화된 방법으로 기술되므로 설계 명세 기법은 ADV_TDS.1와 ADV_TDS.2와 비슷할 것이다.

4. ADV_TDS.4 준정형화된 모듈화 설계 (Semiformal Modular Design)

ADV_TDS.4 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 모듈 단위로 기술함.
- o 서브시스템 설명 기술함.
- o 모듈들이 어떻게 상호작용 하는지 기술함.
- o SFR-수행 모듈과 SFR-지원 모듈은 상세하게 기술함.
  - 기능 설명, 데이터, 인터페이스, 알고리즘 포함
- o SFR-비-간섭 모듈은 기능 설명만 간략히 기술함.
- o 모든 내용은 준정형화된 표현으로 기술됨.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.4 Semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	interaction, purpose

(그림 61) ADV_TDS.4 에 기술되는 정보

ADV_TDS.4 수준에서는 준정형화된 표현 기법을 이용해서 TOE 설계를 명세하므로, 다양한 다이어그램이 사용될 수 있다. 모듈 단위로 기술되는 ADV_TDS.4는 소프트웨어 설계 수준의 명세서와 비슷하며, 설계 명세서에는 기능성을 표현하기 위해 소프트웨어의 정적인 부분과 동적인 부분을 나누어서 기술한다. 소프트웨어 정적인 구조를 보여주기 위해서는 클래스 다이어그램, Structured Chart 등이 사용될 수 있고, 소프트웨어의 정적인 측면을 보여주기 위해서는 시퀀스 다이어그램, 액티비티 다이어그램, DFD 등이 사용될 수 있다. 그리고, 다이어그램과 텍스트 설명이 사용하여 인터페이스 정보를 명세하여 모듈들이 어떻게 상호작용하는지를 기술할 수 있다.

5. ADV_TDS.5 완전한 준정형화된 모듈화 설계 (Complete Semiformal Modular Design)

ADV_TDS.5 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 모듈 단위로 기술함.
- o 서브시스템 설명 기술함.
- o 모듈들이 어떻게 상호작용 하는지 기술함.
- o 모든 모듈은 상세하게 기술함.
  - 기능 설명, 데이터, 인터페이스, 알고리즘 포함
- o 모든 내용은 준정형화된 표현으로 기술됨.

	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.5 Complete semiformal modular design (semiformal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	common data, interfaces, algorithmic

(그림 62) ADV_TDS.5 에 기술되는 정보

ADV_TDS.5는 모든 모듈에 대해 상세하게 기술할 뿐이지 ADV_TDS.4와 명세 방법은 비슷하다. 즉, 다양한 다이어그램과 텍스트 설명을 이용하여 TOE 명세를 기술할 수 있다.

6. ADV_TDS.6 정형화된 상위수준 설계 표현이 제공되는 완전한 준정형화된 모듈화 설계 (Complete Semiformal Modular Design with Formal High-level Design Presentation)

ADV_TDS.6 수준에서 기술되는 TOE 설계 정보는 다음과 같다.

- o 모듈 단위로 기술함.
- o 서브시스템 설명 기술함.
- o 모듈들이 어떻게 상호작용 하는지 기술함.
- o 모든 모듈은 상세하게 기술함.
  - 기능 설명, 데이터, 인터페이스, 알고리즘 포함
- o 모든 내용은 준정형화된 표현과 정형화된 표현으로 기술됨.

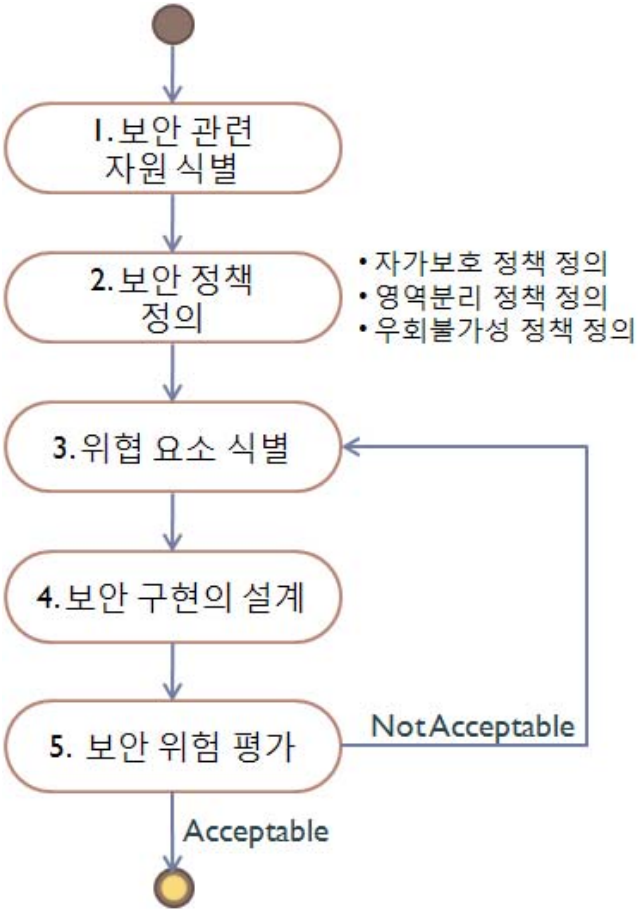
	TSF subsystem			TSF Module		
	SFR Enforce	SFR Support	SFR NI	SFR Enforce	SFR Support	SFR NI
ADV_TDS.6 Complete semiformal modular design with formal high-level design presentation (semiformal presentation; additional formal presentation)	description, interactions	description, interactions	description, interactions	common data, interfaces, algorithmic	common data, interfaces, algorithmic	common data, interfaces, algorithmic

(그림 63) ADV_TDS.6 에 기술되는 정보

ADV_TDS.6는 ADV_TDS.5와 같이 모든 모듈에 대해 상세하게 기술하기 때문에 기술되는 내용은 비슷할 것이다. 그러나, 정형화된 표현을 이용하여 추가적으로 TOE 설계를 설명할 수 있다. Z 명세 언어 등이 이 단계의 컴포넌트를 명세하는데 사용될 수 있다.

제 6 장 ADV_ARC에 특화된 아키텍처 설계 프로세스

본 장에서는 ADV_ARC에 명시되어 있는 세 가지 보안 특성, 즉 자가 보호, 영역 분리, 우회 불가성을 제공하기 위한, 아키텍쳐 설계 프로세스를 제안한다. 또한, 프로세스에 포함된 각 단계 별, 작업 지침(Instruction)을 정의한다. 본 연구에서는 아키텍처 설계를 명세하기 위하여 네 개의 뷰를 사용한다. 기능 뷰 (Functional View)는 목표 시스템의 주요 기능 모듈과 그들 간의 관계를 정의하며, 대부분 아키텍쳐 스타일을 활용하여 명세한다. 정보 뷰 (Information View)는 목표 시스템이 관리, 처리하는 데이터와 데이터간의 관계를 명시한다. 동시 뷰 (Concurrent View)는 목표 시스템 내부에서 실행되는 프로세스나 뜨레드(Thread)가 어떤 순서로, 어떤 동시실행 구조로 처리되는 지를 명시한다. 배치 뷰 (Deployment View)는 대상시스템을 설치하고 운영할 때 필요한 서버의 배치, 모듈의 배치, 네트워킹 등에 관한 결정을 명시한다.



(그림 64) ADV_ARC 기반 아키텍처 설계 프로세스

제안된 설계 프로세스는 그림 64처럼 5개의 단계로 구성된다. 제 1단계, '보안관련 자원식별'은 대상 시스템에서 보안 아키텍처시 관련이 있는 자원(Resource)를 식별하는 단계로서, 이를 기반으로 세 가지 보안 특성이 반영된 아키텍처를 설계하게 된다. 제 2단계 '보안정책(Policy) 정의'는 각 보안 특성별로 적용될 수 있는 보안 정책, 전략, 기법을 정의하는 단계

이다. 이때 정책이란 각 보안 특성을 만족하기 위한, 관리적, 기술적인 기법을 의미하며, 보안 자원의 종류와 보안 특성의 종류에 따라 효과적이며 적합한 정책이 수립되어야 한다. 제 3단계 '위협(Threat) 요소 식별'은 1단계에서 식별된 보안관련 자원에 대해서 3가지 보안 특성의 측면에서 예상되는 위협 요소를 식별하는 단계이다. 보안성이라 위협에 대해서 이겨내는 (Endure)정도 이므로 위협이 식별되어야 정의된 정책을 그 위협에 맞도록 구현하게 된다. 제 4단계 '보안 구현의 설계'는 이미 수립된 보안 정책이 식별된 자원에 적용되도록 구현하는 과정이다. 대부분은 하드웨어나 소프트웨어나 구현하게 되며, 구현 방법은 구현 대상에 따라 상당한 차이가 있다. 제 5단계 '보안 위험 평가'는 설계 및 구현된 보안 정책이 여러 위협에 대비하여 얼마나 안정적으로 보안성을 유지하는 지에 대한 평가이며, 평가 후 위협 수치가 높으면 3단계로 돌아가서 해당 작업들을 다시 수정하여야 한다.

제 1 절 (단계 1) 보안관련 자원 식별

시스템 보안을 설계 하기 전에, 어떤 자원들이 보안을 필요로 하는지의 식별이 되어야 한다. 식별된 자원을 기반으로 하여, 이 후 단계의 지침들이 적용되므로 중요한 작업이다.

- **활동 1-1. 민감한 자원의 식별**

  보안 요구사항을 포함, 기능적 뷰와 정보 뷰를 주로 이용하여, 시스템에서 보안적 측면에서 민감한 자원을 식별한다. 주된 자원으로 기능적 오퍼레이션과 데이터 항목들이 식별된다. 민감한 각 자원에 대해서, 그 자원이 민감한 이유와 누가 그 자원의 소유자이며, 그 자원의 접근 통제 유형을 식별한다. 보안 요구사항이 충분히 상세히 기술되지 않았다면, 관련 당사자와 협의를 통하여 이 자원들을 식별해 낸다.

  보안 관련 자원을 비정형적으로 다양한 방식으로 표현할 수 있지만, 표 5와 같은 형태로 표현할 것을 제안한다.

[표 5] 민감한 자원의 식별의 예제

자원	민감한 정도	소유자	접근 제어
고객 계좌 정보	신원 도용이나 개인 권리 침해의 가능성이 있는 개인 정보	고객 관리 그룹	직접 접근 불가
제품 목록 설명 기재	구매 목록과 관련 설명을 정의한다. 만약 고의적으로 변경을 한다면, 비즈니스에 손상이 있을 수 있다.	주식 관리 그룹	직접 접근 불가
제품 목록 가격 측정 기재	목록 아이템을 위한 가격을 정의한다. 만약 고의적이거나 혹은 우발적으로 변경이 되면, 비즈니스에 손상이나 속임수가 있을 수 있다.	주식 관리 그룹 내에 가격 측정팀	직접 접근 불가
고객 계좌 정보 상의 비즈니스 실행	데이터 접근과 무결성 보호을 위해 통제가 필요하다.	고객 관리 그룹	개별 정보나 혹은 증명된 모든 정보에 접근
목록 운영 기재	데이터 접근과 무결성 보호을 위해 통제가 필요하다.	주식 관리 그룹	증명된 정보로 부터 목록 수정 및 실행하기 위해 접근
가격 목록 수정 실행	데이터 접근과 무결성 보호를 위해 통제가 필요하다.	가격 측정팀	변경된 회계사와 함께 증명된 정보로부터 가격 수정 및 실행하기 위해 접근
...	...	...	...

제 2 절 (단계 2) 보안정책 정의

이 단계는 시스템의 민감한 자원을 식별한 후, 이를 보호하기 위한 보안정책을 수립하여야 한다. 보안정책은 시스템에 보안을 구현하기 위한 기반이 되며, 다음과 같이 정의한다.

> 보안정책이란 누가(Who) 시스템의 어떤 자원을 (Which) 어떤 접근(What)을 할 수 있는 지, 어떤 제약사항이(Constraints) 있는지, 시스템에서 요구되는 무결성(Integrity) 및 자원 접근 시 요구되는 신뢰성(Accountability) 등에 대한 기대치의 명세이다.

이 정의에 따르면, 보안정책은 보안 설계나 구현과 구별되어야 한다. 즉, 보안 정책을 기반으로 구현한 것이 보안 구현이다. 보안 정책을 가능한 정확하게 정교하게 (Precisely) 정의되어야, 해당되는 보안 구현이 효과적으로 실행될 수 있다.

1단계에서 식별된 각 자원별 보안 정책을 수립할 수 도 있지만, 유사하고 관련된 자원들

을 그룹단위로 묶고, 그룹 별 보안 정책을 수립하는 것이 효율적이다.

보안 정책 수립단계는 다음 다섯 개의 활동으로 나누어진다.

- **활동 2-1. 사용자 클래스의 식별**

  보안 측면에서 시스템의 사용자들을 그룹핑하여, 사용자 클래스(Principal Class)를 정의한다. 한 사용자 클래스는 시스템 접근 측면, 사용 자원측면, 사용 형태 측면에서 상당한 유사한 사용자들로 구성된다. 일반적인 그룹핑 기준으로는 사용자 등급 (경영자, 관리자, 운영자, 일반 고객), 지역별 (본사, 지점, 국가), 사업 도메인 별 (공급자, 수요자, 중계자) 등을 들 수 있다.

- **활동 2-2. 자원 클래스의 식별**

  보안에 민감한 자원들을 접근 통제의 관점에서 그룹핑하여 자원 클래스 (Resource Class)를 정의한다. ADV_ARC의 세 가지 특성 중에서 영역 분리와 우회불가성이 밀접한 관련이 있다.

  영역 분리 특성을 지원하기 위해서는 각 영역에는 그 영역에 속한 기능들과 기능 처리에 사용되는 데이터들을 캡슐화하고, 각 영역은 외부 인터페이스에 의해서만 정보를 주고 받을 수 있도록, 정보은닉(Information Hiding) 원칙을 적용하여야 한다. 여기서 영역과 자원클래스는 구별되어야 하는데, 한 영역분리 관점에서의 한 영역에 하나 혹은 그 이상의 자원클래스가 포함될 수 있지만, 한 자원클래스는 한 영역에만 포함되어야 한다. 한 자원클래스가 복수 개의 영역에 포함되어 있을 경우, 영역간의 결합도 (Coupling) 증가로 인해 영역 분리의 원칙을 구현하기 어려워진다.

  우회불가성 특성을 지원하기 위해서는 한 영역에 속한 데이터들이 그 영역의 인터페이스를 통하지 않고는 이 데이터 처리가 불가하도록, 자원을 각 자원클래스에 할당하여야 한다. 특히, 민감한 데이터의 경우는 정보의 가시성을 Private으로 선언하고, 이를 처리하는 오퍼레이션들을 그 종류와 접근 권한으로 통제되도록 자원 클래스를 정의하여야 한다.

- **활동 2-3. 접근 통제 집합의 식별**

  각 자원 클래스에 속한 개별 자원에 적용될 수 있는 오퍼레이션들을 정의하고, 이 오퍼레이션들을 접근해도 좋은 사용자 그룹을 지정한다. 즉, 자원클래스와 사용자 클래스를 접근관계를 정의하고, 이 접근관계의 종류를 명시한다. 이때, 영역분리와 우회불가성의 원칙을 감안하여, 접근 권한을 부여한다.

o **활동 2-4. 민감 오퍼레이션 식별**

시스템 수준에서 보안과 민감한 오퍼레이션들을 식별하고, 이를 접근할 수 있는 사용자 클래스를 정의한다. 예를 들면, 민감한 오퍼레이션으로 시스템 운영, 관리, 사용자 권한 설정 오퍼레이션 등이 있는데, 이들을 사용할 수 있는 사용자 클래스를 정의한다.

o **활동 2-5. 무결성 요구사항 식별**

시스템에서 지워버리거나 수정될 정보를 분석하여, 그 정보의 무결성을 보장할 수 있는 조건이나 보장성에 대하여 명세한다. 즉, ADV_ARC의 세 가지 특성, 각각에 대하여 무결성을 보장할 수 있는 조건이나 목표 시스템, 자원클래스등의 상태를 명시한다.

영역분리 관점의 예로는 한 영역에 속한 자원 클래스가 다른 영역에 속한 민감 오퍼레이션에 의하여 접근되지 않아야 하는 무결성이 있다. 자체보호 관점의 예로는 민감 오퍼레이션의 처리 결과 값이, 정상적인 범위의 값 (Value in Valid Range)을 가져야 하는 무결성이 있다. 우회불가성 관점의 예로는 민감한 자원에 대한 정보 검색 및 수정을 정의된 접근 통제 집합에 명시된 사용자 클래스만 사용하고, 그 자원에 대한 민감한 오퍼레이션 이외는 다른 오퍼레이션이나 수단으로 접근되지 않아야 하는 무결성이 있다.

표 6은 은 위에서 정의한 5개의 활동들을 적용한 적용 예제를 보여준다.

[표 6] 접근 제어 정책의 예제

	사용자 계정 기록	설명서 카탈로그 기록	가격 기록	사용자 계정 조작	설명서 카탈로그 조작	가격 조절 조작
데이터 관리자	모든 기록에 접근 가능 및 평가 가능	모든 기록에 접근 가능 및 평가 가능	모든 기록에 접근 가능 및 평가 가능	모든 조작 가능 및 평가 가능	모든 조작 가능 및 평가 가능	모든 조작 가능 및 평가 가능
카탈로그 사무원	접근 불가	접근 불가	접근 불가	모든 조작 가능	읽기 가능	접근 불가
카탈로그 관리자	접근 불가	접근 불가	접근 불가	평가 및 읽기 가능	모든 조작 가능	모든 조작 가능 및 평가 가능
제품 가격 관리자	접근 불가	접근 불가	접근 불가	접근 불가	읽기 가능	모든 조작 가능
고객 관리 사무원	접근 불가	접근 불가	접근 불가	모든 조작 가능	읽기 가능	접근 불가
등록된 고객	접근 불가	접근 불가	접근 불가	자신의 기록은 모든 조작 가능	읽기 가능	접근 불가
미등록된 웹 사이트 사용자	접근 불가	접근 불가	접근 불가	접근 불가	읽기 가능	접근 불가

제 3 절 (단계 3) 위협 요소 식별

민감한 자원을 등록한 리스트와 보안 정책을 공개 시, 모든 사람들이 준수하는 것은 아니기 때문에 보안 정책을 위협하는 것들을 식별해야 한다. 위협 식별하는 것은 하려면 보호해야 할 시스템이 무엇인지, 무엇으로부터 보호해야 하는지에 관한 명확한 정의를 포함한다. 명확한 정의는 우리가 어떤 위협을 고려해야 하는지 명확하게 해준다.

이 프로세스의 결과는 위협 모델이라 불리고, 이 모델은 민감한 자원의 초기 리스트, 목표하는 시스템에서 발생 가능한 위협 분석, 위협 발생 시 영향, 위협 발생 가능성 등을 포함한다.

위협 모델을 생성하기 위하여, 제안된 시스템에 다음과 같은 주요 질문을 할 수 있다.

- 보안 정책을 위반할 수 있는 사람은 누구인가?
- 어떻게 위반할 수 있는가?
- 침입자들의 주요 특징(정교함, 동기, 접근하려고 하는 자원 등의 정보)은 무엇인가?
- 정책 위반 시에 결과는 무엇인가?

위협의 명시적 식별은 프로젝트 외부 전문인이 위협 모델을 검사하고 잠재적인 위협에 대해 지원할 수 있게 한다. 잠재적인 위협으로는 전혀 고려되지 않은 위협이나 위협 모델에 잘못 특징화된 위협이 될 수 있다. 위협 모델을 전반적으로 검사하여 이 모델을 확실하게 함으로써 가능한 위협을 모두 고려할 수 있으며 시스템에서 필요한 보안 장치를 고려할 수 있다.

위협 요소 식별은 다음 세 개의 활동으로 나누어진다.

o **활동 3-1. 위협 식별**

시스템의 보안에 위협이 되는 것을 식별한다. 위협 식별 방법은 다음 세 개의 속성 별로 나누어 진다.

- 자체 보호

과거 기록을 이용하여 자원의 침입 예상 가능성 및 관심도를 식별하고 자원의 민감한 정도에 따라 침입 시 예상 위험을 식별한다. 또한, 잠재적인 침입자의 주요 특성, 목표, 동기, 예상 침입 방법, 침입의 종류 등을 식별한다. 특히, 예상 침입 방법의 경우, 트리로 구성하는 것은 유용한 방법이다. 그림 65는 예상 침입 방법을 글자 형태의 트리로 표현한 예를 보여주고 있다.

목표: 고객 카드 정보 획득

1. 시스템 데이터 베이스로부터 자세한 정보 획득
   - I. 데이터 베이스 직접 접근
     - I. 데이터 베이스 암호 해독/추측
     - II. 운영 시스템 암호 해독/추측
   - II. 데이터 베이스 관리자 멤버를 통한 접근
     - I. 데이터 베이스 관리자 매수
     - II. 데이터 베이스 관리자 속임

2. 웹 인터페이스로부터 자세한 정보 획득
   - I. 가짜 웹 사이이트를 설치, 사용자에게 이메일로 전송하여, 신용 카드 정보를 입력하게 속임
   - II. 사용자 계정 암호 해독/추측 및 사용자 웹 인터페이스를 통한 정보 추출
   - III. 이메일을 통해 트로잔 프로그램을 사용자에게 전송하여, 사용자 키 입력을 기록
   - IV. 사이트 서버 소프트웨어에 직접 침입하여 보안에서 허술한 부분 획득

(그림 65) 고객 카드 정보 예상 침입 방법 트리

- 영역 분리

영역 간의 고유 식별자, 각 영역에 접근 가능한 객체의 식별자를 식별하여, 영역 간

의 의존성을 식별한다. 또한, 다른 영역에 접근 가능한 객체의 주요 특성, 목표, 의도, 접근의 종류 등을 식별한다. 예상치 못한 영역이 새롭게 정의됨으로써 영역 간의 결합도가 증가하는 경우, 영역 간의 호출 경로 및 방법을 식별한다. 이와 같은 호출 경로는 그래프를 이용하여 표현할 수 있다.

- 우회 불가성

TSF 인터페이스 사용 대비 TOE 인터페이스 사용 횟수, TOE에 정의된 public 데이터 및 인터페이스를 식별한다. 또한, TOE 인터페이스 사용 경로를 식별하여, TOE 인터페이스의 직접 사용 및 TSF 인터페이스를 이용한 TOE 인터페이스 사용을 식별한다. 또한, 이를 기반으로 인터페이스들의 응집력 및 결합도를 식별한다.

o **활동 3-2. 위협 특성화**

위협의 해결 방법, 해결한 후의 결과, 위협 발생 가능성 관점에서 각 위협을 특성화한다. 위협 특성화 방법은 다음 세 개의 속성 별로 나누어 진다.

- 자체 보호

민감한 자원의 침입이 성공하였을 시 발생 위험도 및 침입의 종류에 따라 해결 방법을 고려한다. 또한, 침입자의 목표를 해결 방법에 반영하여야 한다. 해결한 후의 결과는 침입에 따른 자원의 손실 정도, 손실로 인하여 시스템이 영향 받은 정도 등을 기반으로 도출해야 한다. 자체 보호 위협 발생 가능성 항목에서는 침입 예상 가능성, 민감한 자원의 관심도, 침입 성공 가능성 등을 고려한다.

- 영역 분리

허용되지 않은 객체의 접근이 성공하였을 시 영역 간의 결합도 증가 정도에 따라 해결 방법을 고려한다. 또한, 새로운 영역이 정의됨으로써 영역 간의 의존성이 발생하였을 시 영역 간의 결합도 발생에 따라 해결 방법을 고려한다. 해결한 후의 결과는 결합도 증가에 따른 보안 손상도, 손상으로 인한 시스템 보안의 복잡도 증가 등을 기반으로 도출한다. 영역 분리 발생 가능성 항목에서는, 허용되지 않은 객체의 접근 가능성, 접근 성공 가능성, 새로운 영역 정의시 결합도 증가 정도 및 결합 가능성 등을 고려한다.

- 우회 불가성

TOE의 public 인터페이스를 이용하여 기능을 직접 사용하였을 시 사용 경로 재설정을 통한 해결 방법을 고려한다. 또한, TOE와 TSF 인터페이스의 응집력 감소시 인터페이스들 재정의를 통한 해결 방법을 고려한다. 해결한 후의 결과는 직접 사용에 따

른 위험 발생도, 위험으로 인한 시스템 보안 손상 정도 등을 기반으로 도출한다. 우회 불가성 발생 가능성 항목에는, TOE의 직접 사용 확률 및 가능성, TOE와 TSF 인터페이스의 응집력 감소 가능성 및 감소 정도 등을 고려한다.

제 4 절 (단계 4) 보안 구현의 설계

민감한 자원과 위협을 기반으로, 시스템을 위한 기술적 보안 설계를 한다. 목표는 시스템 전반적인 보안 기반 구조를 설계하고, 위협 모델에서 식별된 위험을 해결하기 위한 시스템 보안 정책을 강화이다. 보안 설계 시 통합 인증(Single-sign-on) 시스템, 네트워크 방화벽(Network firewalls), SSL 통신 연결 보안, 암호화 기술, 정책 관리 시스템과 같은 특정 보안 기술을 사용할 수 있다.

설계 프로세스를 통해 아키텍처와 여러 가지 설계 결정 사항이 도출된다. 이런 결정 사항은 기능, 정보, 동시, 배치 뷰에서 기술된 아키텍처 구조에 반영된다.

보안 설계의 구현은 다음 네 개의 활동으로 나누어 진다.

o **활동 4-1. 위협 완화 방법 설계**

식별된 위협마다 처리할 수 있는 보안 메카니즘을 설계한다. 설계 방법에는 기존 설계 결정 사항 수정, 하나 이상의 보안 기술 적용, 시스템 운영 및 사용을 위한 절차 및 프로세스 설계 등이 포함된다. 위협 완화 방법은 다음 세 개의 속성 별로 나누어 진다.

- 자체 보호

민감한 자원의 접근 관리 구현을 설계하기 위해서는, 디자인 패턴에서 제안되어 있는 proxy 패턴을 이용하여, 민감한 자원을 직접 접근하지 않도록 해야 하며, 또한, 접근 요청 유효성을 확인할 수 있는 알고리즘을 사용하여야 한다. 침입자의 접근 특성에 따른 인증 방법 설계, 침입 방법에 따른 필터링 방법 설계 등이 필요하다. SSL 통신 연결 보안 등을 적용할 수 있다.

- 영역 분리

영역 간 인증, 접근의 무결성 확인 알고리즘 설계, 특성에 따른 객체 접근 제한 방법 설계 등이 필요하다. 암호화 기술 등을 적용할 수 있다.

- 우회 불가성

TOE의 public 인터페이스 설계 수정, TOE와 TSF 인터페이스의 응집력 강화 방법, TSF의 public 인터페이스 공개 및 TOE의 인터페이스 은닉을 통한 시스템 운영 방법

등이 필요하다.

우회불가의 무결성의 구현을 설계하기 위해서는, Java언어에서 해당되는 민감 오퍼레이션을 Final로 선언하여, 서브 클래스에서 우회 접근되지 않도록 해야 하며, 또한, 자원을 Private으로 선언하여 외부에서의 가시성을 차단하여야 한다.

o **활동 4-2. 발견 및 회복 방법 설계**

시스템 전반적인 측면에서 시스템 보안 정책 위반을 발견하고 회복하는 방법을 설계한다. 발견 및 회복 방법 설계는 다음 세 개의 속성 별로 나누어 진다.

- 자가 보호

침입 발견 방법, 침입의 정기적인 발견과 발견에 따른 반응을 위한 일련의 프로세스를 포함하고 있다. 상업용 혹은 오픈 소스 소프트웨어를 이용하여 침입을 발견할 수 있으나, 소프트웨어의 사용이나 프로세스 설계는 시스템 특화된 활동이다.

- 영역 분리

객체 접근으로 인한 예상치 못한 불일치의 확인 및 조절, 새롭게 정의된 영역으로 인한 예상치 못한 결합도 증가의 확인과 감소를 위한 일련의 프로세스를 포함하고 있다.

- 우회 불가성

TOE 기능 호출 흐름 발견 방법, 직접적인 기능 호출의 정기적인 확인과 발견시 TSF 인터페이스를 통한 TOE 기능 사용을 위한 경로 설정 방법을 위한 일련의 프로세스를 포함하고 있다. 이를 위해서는 호출 경로를 수정할 수 있는 중재 모듈이 필요하다.

o **활동 4-3. 기술 평가**

위협을 다루는 방법 중 하나는 보안 메커니즘을 제공하는 보안 기술을 사용하는 것이다. 그러기 위해서는 특정 문맥에서 특정 위협을 다루기에 알맞은 후보 보안 기술을 평가하여야 한다. 평가 방법은 신뢰성 확인, 사용성 등의 일반적인 평가 방법과 제안된 기술의 기능과 시스템 성능 및 규모와의 적합성 확인 등의 문맥 특화 평가 방법으로 구성된다. 문맥 특화 평가 방법은 다음 세 개의 속성 별로 평가 항목이 세분화 된다.

- 자가 보호

침입 발견의 신뢰도, 민감한 자원 접근 제한 성능 등

- 영역 분리

  영역 무결성 확인 신뢰도, 영역의 응집력 및 결합도 평가 적절성 등

- 우회 불가성

  TOE 공용 인터페이스의 사용 범위, 기능 호출 흐름 확인 신뢰도, 경로 재설정의 효율 및 신뢰도

o **활동 4-4. 기술 통합**

기본 시스템 구조와 보안 기술을 통합하는 방법을 결정한다. 단순 통합으로 인한 부작용으로 인해 가능한 보안 문제를 피하기 위해 통합 방법 설계는 신중히 해야 한다. 자가 보호, 영역 분리, 우회 불가성에서 발생할 수 있는 위협을 실시간으로 점검할 수 있는 모듈 및 문제 발견 시 반응할 수 있는 모듈을 추가 시 세 가지 속성을 해치지 않도록 해야 한다.

제 5 절 (단계 5) 보안 위험 평가

보안이 제공되지 않는 시스템은 완벽하지 않다. 시스템 보안을 구현하는 프로세스는 균형을 조절하는 활동이다. 즉, 위험이 발생하였을 때 소요될 수 있는 비용과 예측된 위험의 해결책을 구현하는데 소요되는 비용을 고려하여 균형을 조절한다.

시스템을 위한 보안 기반 구조를 설계 후에, 위험과 비용 간의 균형 조절의 적절성을 고려하여 위험을 다시 재평가할 필요가 있다. 만약, 균형 조절이 적절했다면, 보안 관점은 완성이 된 것이다. 그렇지만, 균형 조절이 적절하지 않았다면, 위협 모델과 보안 기반 구조를 정제하여, 위험과 비용 간의 균형 조절을 달성할 수 있도록 한다.

o **활동 5-1. 위험 평가**

보안 위험 평가의 유일한 활동은 위험 평가이다. 위험 평가를 위해서는 다음과 같은 것들을 고려한다.

- 자가 보호

  민감한 자원의 접근 가능성, 잠재적인 침입의 가능성 등을 재평가하고, 재평가 결과 및 자원의 민감한 정도에 따라 보안성 필요 정도를 조절한다. 또한, 침입에 따른 보안 위험이 적정 수준으로 정의되었는지 평가한다.

- 영역 분리

  영역 분리에서는 예상치 못한 불일치의 비율, 새로운 영역 정의로 인한 결합도 증가

가능성, 객체의 접근 가능성 등을 재평가하고, 재평가 결과 및 영역의 중요도에 따라 보안성 필요 정도를 조절한다. 또한, 영역의 결합도 증가에 따른 보한 위험이 적정 수준으로 정의되었는지 평가한다.

- 우회 불가성

TOE의 기능 직접 사용 확률, TOE와 TSF 인터페이스의 기능 및 구조적 응집도 등을 평가하고, 재평가 결과 및 TOE의 영향력에 따라 보안성 필요 정도를 조절한다. 또한, TOE 직접 이용에 따른 보안 위험이 적정 수준으로 정의되었는지 평가한다.

제 7 장 아키텍처 설계 명세 기법

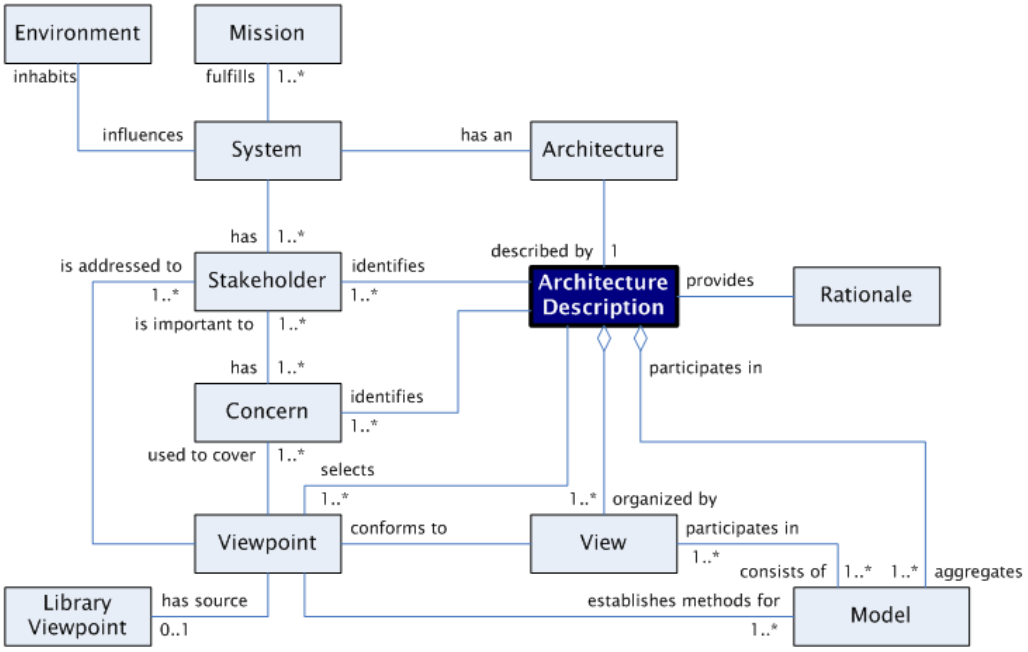
보안 아키텍처를 평가하기 위해서는 설계자 또는 아키텍처 설계자가 해당 TOE에 대해 설계한 아키텍처에 대한 명세서가 필요하다. 그러므로, 본 장에서는 아키텍처를 명세하기 위한 표준인 IEEE P1471과 SEI(Software Engineering Institute)에서 제안한 아키텍처 명세 기법을 조사하고, 이를 기반으로 보안 요구사항에 적합한 아키텍처 명세서를 제안한다. 그리고, TOE 설계(ADV_TDS), 보안구조(ADV_ARC) 요구사항과 관련한 아키텍처를 평가하기 위한 아키텍처 명세서(Architectural Description) 작성에 사용될 수 있는 지침을 개발한다. 이 지침은 아키텍처 설계와 소프트웨어 설계에 관한 범용적인 지침과 양식(Template)을 포함한다.

제 1 절 기존 아키텍처 명세서

1. IEEE P1471

IEEE P1471은 “IEEE Recommended Practice for Architectural Description of Software-Intensive Systems”로서, 아키텍처와 관련된 용어와 개념을 통일하며, 아키텍처 명세서에 대한 권장사항(recommendation)을 제시한다 [8]. 제안된 아키텍처 명세서는 정해진 모델링 언어, 방법론 같은 것을 제시하지 않기 때문에 기술에 중립적이며, 프로젝트 규모에 따라 유연하게 적용할 수 있다.

그림 66는 IEEE P1471에서 제안한 아키텍처 명세서에 대한 메타모델이다.



(그림 66) IEEE P1471의 아키텍처 명세에 관한 개념적인 모델

IEEE P1471은 아키텍처와 관련된 다양한 항목과의 연관 관계를 다음과 같이 설명한다.

o 시스템과 아키텍처와의 관계

모든 시스템은 아키텍처를 가진다. 아키텍처는 모든 시스템에 내재된 것이기 때문에 아키텍처를 생각하지 않고 만든 시스템이더라도 아키텍처를 가지고 있다. 다만 아키텍처의 질(quality)이 문제가 될 뿐이다. 제대로 된 아키텍처라면 특정 환경의 영향을 받는 시스템이 이해 당사자들의 관심을 조절해서 비즈니스 목적을 달성할 수 있어야 한다.

- 시스템(System)은 비즈니스 목적이나 사명(Mission)을 완수해야 한다.
- 시스템은 특정 환경(Environment)의 영향을 받으면서 개발되고 운영된다.
- 시스템은 여러 이해 당사자(Stakeholder)와 연관이 있다. 이해 당사자들이 시스템의 사명과 환경을 결정한다.
- 모든 시스템은 아키텍처(Architecture)를 하나 가진다.

o 이해 당사자와 아키텍처 명세서의 관계

- 이해 당사자는 시스템에 관심(Concern)을 갖는 사람이나 조직이다.
- 이해 당사자들의 관심은 아키텍처로 수렴된다. 따라서, 아키텍처는 여러 이해 당사자들의 관심을 조율하고 절충해서 만족시켜야 한다. 이해 당사자들은 아키텍처를 활용해서 상호작용하고 자신의 관심을 만족시킨다. 따라서, 아키텍처 명세서를 작성할 때 이해 당사자와 이해 당사자의 관심을 식별하는 것은 아주 중요하다.

o 아키텍처 명세서와 아키텍처의 관계
- 아키텍처는 아키텍처 명세서로 문서화되어야 활용될 수 있다.
- 아키텍처 명세서는 아키텍처를 결정한 근거(Rationale)를 제시해서 쓸 데 없는 논쟁이나 자원 낭비를 막아야 한다.

o IEEE P1471에서 정의한 아키텍처 명세서
- 이해 당사자와 이해 당사자들의 관심을 식별해서 아키텍처 명세서에 명시해야 한다.
- 아키텍처 명세서는 뷰(view)들로 이루어진다. 뷰는 이해 당사자들의 시스템에 대한 관심을 뷰포인트(viewpoint)에 맞춰 작성한 것이다.
- 뷰포인트는 모델(model) 작성 방법을 정의하고 어떤 모델(Model)의 어떤 부분이 어떤 뷰을 만들 때 중요하고 꼭 필요한 것인지 어떤 것이 중요하고 꼭 필요한지 정의한다.
- 뷰포인트의 정의에 따라 모델(model)들 가운에 중요하고 꼭 필요한 부분을 모아서 뷰를 만든다.
- 이해 당사자의 관심을 만족시킬 수 있도록 뷰포인트를 선택해야 한다.
- 뷰포인트는 새로 만들 수도 있고 미리 정의된 뷰포인트인 라이브러리 뷰포인트(library viewpoint)를 활용할 수도 있다.

그리고, IEEE P1471에서는 아키텍처 명세서를 작성하는 절차를 다음과 같이 나열한다.

o 1. 아키텍처 명세서 정보를 작성한다.

아키텍처 명세서 전체에 대한 전반적인 정보를 작성한다. 예를 들어, 작성일자, 변경이력, 요약, 문서의 범위, 용어 정리, 참고 사항 등이 이에 포함된다.

o 2. 이해 당사자와 이해 당사자별 관심을 식별한다.
- 사용자는 주로 시스템이 제공해주는 기능에 관심을 가진다.
- 인수자는 주로 시스템이 비즈니스 목표를 완수하는지, 투자대비 효과는 좋은지, 시스템의 상품성은 있는지 같은 시스템의 경제성에 관심을 가진다.
- 개발자는 주로 시스템 구현과 관련된 기술 분야에 관심을 가진다.
- 유지보수자는 완성된 시스템의 확장이나 변경에 관심을 가진다.

o 3. 뷰포인트(Viewpoint)를 선택한다.

뷰포인트는 뷰를 작성하는 규칙과 방법을 정해 놓은 뷰의 메타모델로서, 뷰를 구성하는 모델을 정의한다. 모델 종류 뿐만 아니라 모델 작성 언어, 모델 작성 방법, 분석 기법도 정의한다. 다양한 이해 당사자들은 자신이 관심을 가지고 있고 잘 아는 부분을 바라본다. 이렇게 이해 당사자들이 바라보는 부분을 뷰로 모으면 아키텍처를 설명할

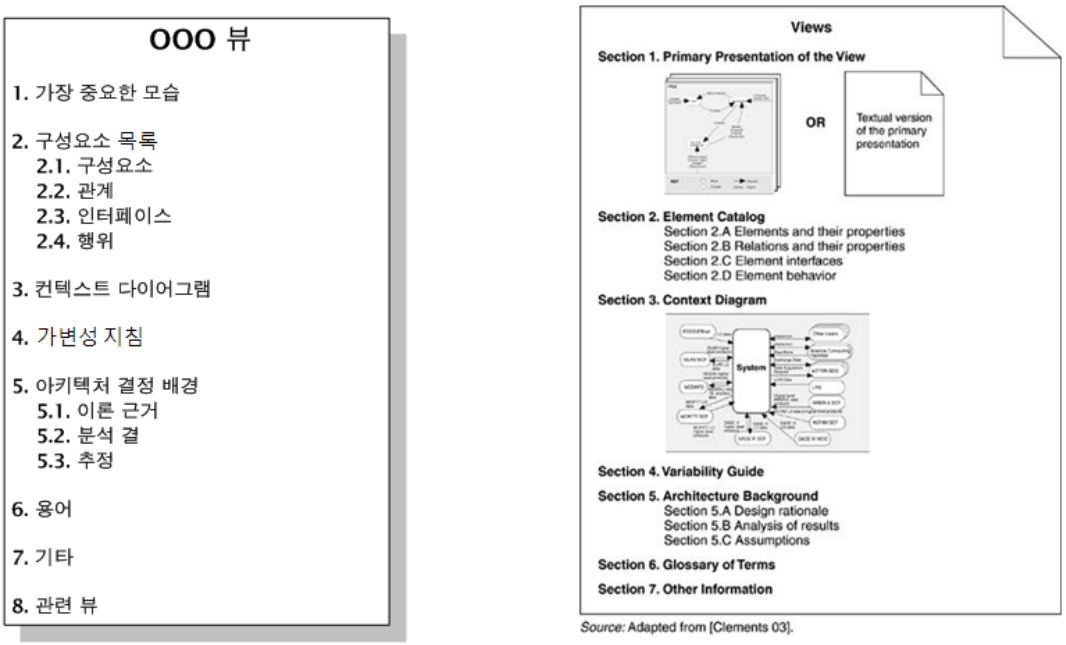
수 있다.

o 4. 뷰포인트별 설명을 작성한다.
아키텍처 명세서에는 적용하기로 한 뷰포인트에 대한 설명이 필요하다. 기술되는 항목은 뷰포인트 이름, 해당 뷰포인트와 관련 있는 이해 당사자와 이해 당사자의 관심 사항, 뷰를 만드는 방법, 해당 뷰포인트를 선택한 근거(rationale)이 있다.

o 5. 뷰를 작성한다.
뷰포인트에 맞춰 뷰를 작성하며, 우선순위가 높은 뷰포인트부터 뷰를 만든다. 이 단계에서는 결정된 뷰포인트에서 정의한 방법에 따라 뷰를 구성할 모델을 선정한다. IEEE P1471에서는 어떤 항목으로 뷰를 기술해야 하는지는 구체적으로 나열하지 않았다.

2. SEI 아키텍처 명세 기법

SEI에서는 아키텍처 명세서의 중요성을 강조하면서, 이와 관한 책[3] 등을 출판함으로써 아키텍처 명세서의 템플릿과 포함되어야 하는 내용을 설명한다. 아키텍처 명세서는 새롭게 회사에 입사한 직원들을 교육시키는데 사용되는 자료도 활용될 수 있으며, 이해 당사자 간의 의사소통을 원활히 하는데 보조역할을 하며, 개발되는 시스템을 분석하는데 기초적인 자료를 제공한다.



(그림 67) SEI에서 제시한 아키텍처 명세서의 뷰 명세 구조

o 가장 중요한 모습(Primary Presentation)

이해 당사자가 한 눈에 뷰를 파악할 수 있도록 뷰를 구성하는 구성요소와 구성요소들 사이의 관계 가운데 제일 중요한 것을 추려서 표현한 것이다.

대부분 그림과 그림에 대한 설명으로 작성한다. 간단한 문장이나 표로 작성하는 경우도 있다.

o 구성요소 목록(Element Catalog)

가능하다면 뷰의 모든 구성요소들과 관계를 자세히 설명한 목록을 만드는 것이 좋다. 하지만, 상황이 어렵다면 가장 중요한 모습에 나온 구성요소와 관계만이라도 설명해야 한다. 구성요소와 관계만으로 부족하다면 구성요소들의 행동을 설명한 행위와 상호작용 방식을 정의한 인터페이스도 설명해야 한다.

o 컨텍스트 다이어그램(Context Diagram)

현재 뷰에서 논의하고 있는 시스템이나 시스템의 한 부분이 외부 환경과 어떤 관계를 맺고 있는지 보여준다.

o 가변성 지침(Variability Guide)

아키텍처의 어떤 부분은 결정을 내릴 수 없고 선택사항만 있는 경우가 있다.

이런 부분은 가변성 지침에 명기하고 선택사항을 고르는 방법도 설명해야 한다. 하나를 선택해야 하는 경우에 어떻게 선택해야 하는 지 설명한다.

o 아키텍처 결정 배경(Architecture Background)

뷰를 구성하는 모델들의 정당성을 제공한다.

- 이론 근거 : 다음 아키텍처 설계자가 쓸데 없는 노력을 하지 않도록 여러 가지 설계안 가운데 현재 설계안을 선택한 이유와 다른 안을 버린 이유를 설명한다.
- 분석결과 : 결정을 내리기 위해 분석을 했다면 분석결과를 제공하여 근거자료로 삼는다.
- 추정 : 결정을 내릴 때 당연한 것으로 추정한 것이 있다면 밝혀야 한다. 보통 주변 환경이나 예상되는 요구를 보통 가정한다.
-

o 용어 정리(Glossary of Terms)

뷰에 나온 용어들을 간략하게 정의한다.

o 기타 정보(Other Information)

저자, 형상관리, 변경관리 같은 관리 정보나 아키텍트가 특별히 언급하고 싶은 요구사

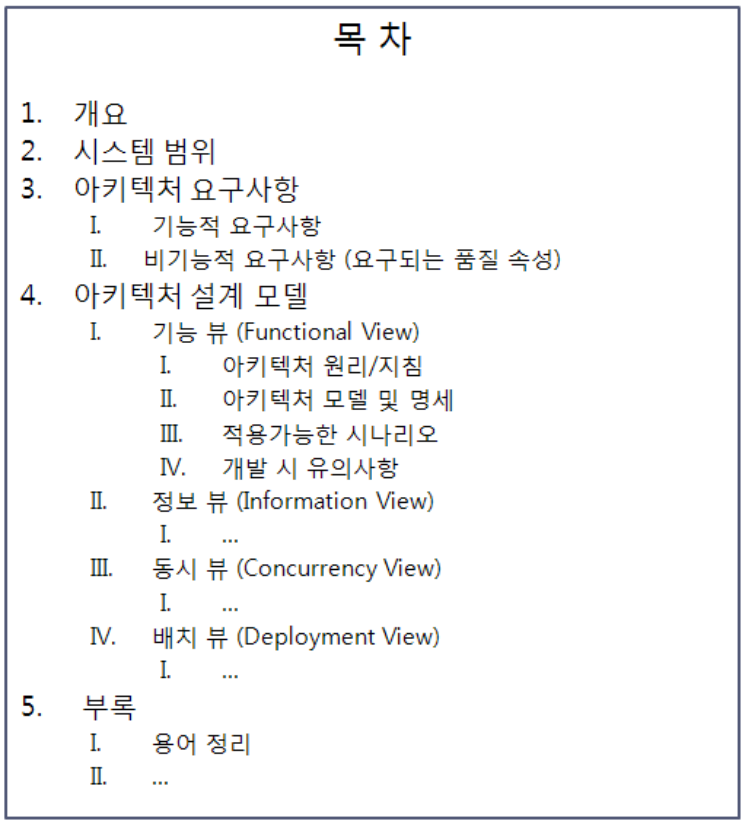
항과 추적성을 기록한다.

o 관련 뷰(Related View)

제 2 절 아키텍처 설계 명세서 작성 기법

아키텍처 명세서(Architectural Description)는 이해당사자들이 아키텍처가 이해당사자의 요구사항 또는 관심사항을 모두 반영했다는 것을 이해하고 증명할 수 있도록 아키텍처를 명세함으로써 나오는 산출물이다. 아키텍처 명세서의 목적은 개발되는 소프트웨어 또는 시스템의 연관된 모든 이해당사자들과 함께 의사소통하기 위함이고, 전체 시스템의 기능적 요구사항과 비기능적 요구사항(품질 요구사항)에 대한 이해도를 구축하는데 도움이 되며, 아키텍처 관심사항(architectural concern)을 해결하기 위해 올바른 결정을 하였다는 것을 확실하게 해준다.

위와 같은 목적으로 아키텍처 명세서가 기술되기 때문에, 아키텍처 명세서는 이해 당사자의 요구사항과 관심 사항을 정확하게 나타내어야 하며, 이해당사자의 요구사항을 정확히 만족시키는 아키텍처를 명세해야 한다. 그리고, 이해당사자가 아키텍처에 관한 질문을 할 때, 충분히 대답을 할 수 있을 정도로 충분하게 기술되어야 한다. 그러나, 아키텍처 명세서는 이해당사자의 관심사항과 시스템에 대한 이해도, 수행되는 시스템에서 해결해야 하는 문제점의 복잡도, 소프트웨어 설계자의 역량, 보증 정도 등에 따라 다양한 형태로 기술될 수 있기 때문에 범용적인 아키텍처 명세서의 템플릿을 작성하는 것은 어렵다. 그러므로, 본 절에서는 보안 아키텍처를 명세하기 위한 목차를 제안하고, 각 목차별 내용에 대한 가이드라인은 TDS 계층 구조 별로 설명한다.



(그림 68) 보안 아키텍처 명세서 목차

그림 68는 ADV_ARC를 위해 설계된 아키텍처를 위한 명세서의 목차를 보여준다. 아키텍처 명세서는 총 5개의 장으로 구성되며, 아키텍처 명세서를 읽는 이해 당사자들의 이해를 돕기 위해 각 장마다 간략하게 장의 목적과 주요 내용을 기술하는 것이 좋다.

다음은 각 장 별로 어떤 내용이 기술되어야 하는지를 보여준다.

o **"1장. 개요" 작성 지침**

이 장에서는 명세될 아키텍처 명세서를 이해하는데 도움이 될 수 있는 시스템에 대한 개략적인 설명을 기술한다. 다음은 이 장에 포함될 수 있는 명세 항목이다.

- 아키텍처 명세서 목적
- 명세될 소프트웨어 아키텍처가 포함되는 시스템의 목적에 대한 요약
- 중요 요구사항과 시스템 기능 범위에 대한 요약
- 중요 요구사항을 아키텍처에서 어떻게 해결할 수 있는지에 대한 개략적인 설명
- 관련 이해 당사자에 대한 요약

1장에 기술되는 내용은 명세서를 읽을 이해당사자의 시스템 또는 아키텍처에 대한 이해 정도에 따라 기술되는 내용의 깊이가 다를 수 있다. 즉, 아키텍처 명세서를 읽을 사람이 시스템 전문가일 경우에는 1장에 많은 내용이 기술되지 않을 수 있다. 반면에, 시스템에 대해 많이 알지 못하거나 아키텍처에 대해 생소한 사람이 이 명세서를 읽게 된

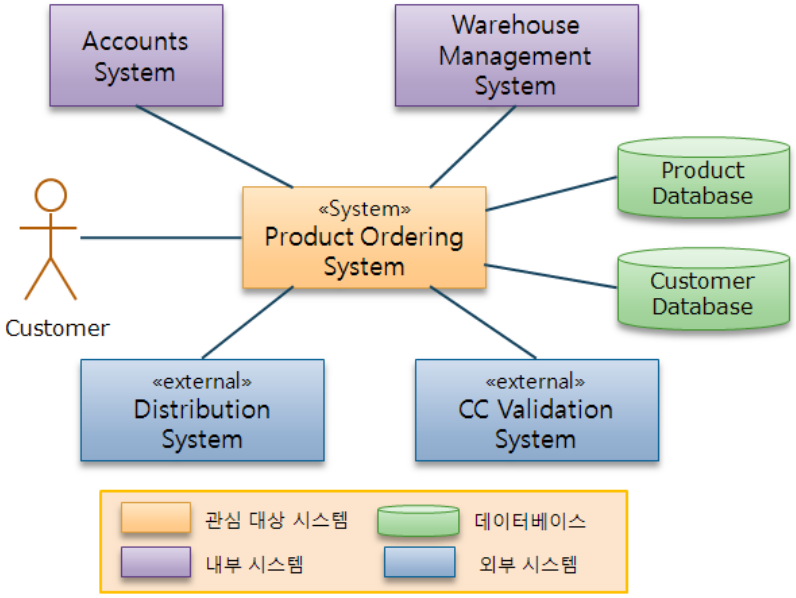
다면, 1장에 보다 상세한 내용이 기술되어야 한다.

o **"2장. 시스템 범위" 작성 지침**

이 장은 개발되는 대상인 시스템에 대한 범위 설명, 시스템을 사용하는 다양한 종류의 사용자, 시스템과 연동하는 외부 시스템에 대한 정보를 포함한다. 아키텍처 명세서를 작성하기 이전에 이와 관련된 다른 문서가 있다면, 그 문서를 참조하면 되고, 그렇지 않은 경우에는 다음과 같은 항목을 포함하여 작성하면 된다.

- 시스템에 의해 제공되는 기능의 범위
  (즉, 개발되는 시스템이 제공할 수 있는 기능의 범위)
- 만약 개발되는 시스템이 복잡할 경우, 시스템의 컨텍스트 정보도 포함시킴.
  - 시스템과 상호작용하는 외부 시스템
  - 외부 시스템의 외부 인터페이스
  - 개발되는 시스템의 인터페이스
    (이 인터페이스는 외부 시스템과 연관된 인터페이스임)

    2장의 내용을 기술할 때에는 필요에 따라서, 컨텍스트 다이어그램이 유용하게 사용될 수 있다. 컨텍스트 다이어그램은 시스템의 경계와 시스템과 상호작용하는 외부 개체((Entity, 외부 시스템 또는 외부 액터)를 상위 수준에서 기술한 다이어그램이다. 컨텍스트 다이어그램을 그림으로써, 시스템에서 제공되어야 하는 기능을 분명히 할 수 있다. ADV_TDS 의 4와 5단계에서 준정형화된 표현을 이용하여 아키텍처 명세서를 기술할 경우, 컨텍스트 다이어그램은 UML 표준으로는 유즈케이스 다이어그램이 이용될 수 있고, 그림69과 같은 형태로 표현할 수 있다. 표준을 사용하지 않을 경우에는, 다이어그램에 범례를 명시하여 각 표기법이 가지는 의미를 기술해야 한다. 그림69은 UML의 스테레오 타입을 이용하여 관심 대상이 되는 시스템은 《System》을 이용하여 표기하였고, 시스템과 상호작용 하는 시스템은 《external》을 이용하여 나타내었다. 그리고, 개발되는 시스템을 사용하는 다양한 종류의 사용자는 유즈케이스 다이어그램의 액터 표기법을 이용하여 나타내었다.



(그림 69) 컨텍스트 다이어그램의 예

o **"3장. 아키텍처 요구사항" 작성 지침**

만약 아키텍처 요구사항을 별도의 문서에 기술하였다면, 이 장에서는 기존에 작성한 요구사항 명세서(Software Requirement Specificatin)을 참조하면 된다. 만약, 요구사하에 대해 별도로 기술한 문서가 없다면, 이 장에는 다음과 같은 내용이 포함된다.

- 기능적 요구사항(Functional Requirement)

  시스템이 해야 할 일을 나타내는 기능적 요구사항을 기술한다. 가능하다면, 각 기능적 요구사항은 텍스트 형식으로 기술되기보다는, 항목별로 분류하고 각 항목에는 번호를 부여하며, 관련된 요구사항은 그룹화하여 기술하는게 요구사항을 이해하는데 도움이 된다. 여기에서는 시스템의 모든 기능적 요구사항을 상세기 기술하기보다 시스템을 이해하는데 충분한 정도의 내용으로 기술한다.

- 비기능적 요구사항(Non-functional Requirement, Extra-functional Requirement)

  시스템이 해야하는 기능적 요구사항은 아니지만, 시스템이 어떻게 작업을 수행해야 하는지와 관련된 요구사항을 기술한다. 대표적으로는 품질 요구사항이 기술되며, 그 예로는 성능(performance), 이용가능성(availability), 확장 가능성(scalability), 보안성(security) 등이 있다. 기능적 요구사항과 마찬가지로, 비기능적 요구사항도 항목별로 분류하고 각 항목에 번호를 부여하여 기술하는 것이 좋다. 아키텍처를 설계하는 데에는 기능적 요구사항보다는 비기능적 요구사항이 영향을 더 많이 미치므로, 비기능적 요구사항을 아키텍처 요구사항(Architectural requirement)라고도 부른다.

- 아키텍처 요구사항의 중요도 (선택적)

시스템의 이해당사자는 각기 다른 관심 사항을 가지고 있을 수 있다. 한 시스템에 다양한 이해당사자가 관련되어 있을 경우, 이해당사자들의 각각 다른 관심사항으로 아키텍처 설계 또는 시스템 분석/설계가 어려워질 수 있다. 그러므로, 각 이해당사자별 아키텍처 요구사항을 분석하고, 가장 중요한 아키텍처 요구사항(First-cut concerns)이 무엇인지를 찾아내는 작업 또는 중요하다.

o **"4장. 아키텍처 설계 모델" 작성 지침**

이 장이 아키텍처 명세서에서 가장 중요한 부분으로, 개발될 시스템에 포함되는 아키텍처가 어떻게 설계되었는지를 기술한다. 아키텍처 명세서를 통하여, 모든 이해당사자들이 가지는 다양한 관심에 따라 표현해야 하는 방법도 다양하다. 그러므로, 텍스트 형태, 다이어그램, 또는 아키텍처 명세 언어(Architecture Description Language, ADL)등의 다양한 방법을 이용하여 기술한다. 학계에서는 ADL을 개발하고 있지만, 산업계에서는 UML을 아키텍처 명세 언어로 널리 사용하고 있다.

아키텍처 설계를 뷰별로 작성해야 하며, 보안 아키텍처는 앞서 6장에서 언급하였듯이 기능 뷰, 정보 뷰, 동시 뷰, 배치 뷰로 나누어 아키텍처 설계 모델을 구체적으로 기술한다.

- 아키텍처 설계 원리

아키텍처 설계 원리는 아키텍처 설계를 뒷받침하는 설계 의도, 설계 접근 방법 등에 대한 간략한 설명을 기술한 것으로, 현재 상황(as-is)와 요구되는 미래 상황(to-be)을 참고하여 설명할 수도 있다. 아키텍처 설계 원리는 아키텍처를 정의하는데 필요한 기준선을 확립하는데 중요한 역할을 하며, 이해당사자의 가정 또는 이해당사자의 개발될 시스템 또는 아키텍처에 대한 기대치 등을 포함하기도 한다. 즉, 아키텍처 설계 원리에는 설계된 아키텍처의 해당 뷰에서 반드시 해결되어야 하는 아키텍처 요구사항들을 기술한 내용과 비슷하게 기술될 것이다. 아키텍처 설계 원리를 기반으로 아키텍처 설계에 중요한 영향을 미치는 결정사항(Architectural Decision)이 이루어진다. 그리고 이런 결정사항은 아키텍처 모델에 표현된다.

아키텍처 설계 원리의 예로는 "모든 사용자는 하나의 접근점(single access point)를 통하여 정보, 서비스 등을 접근해야 하며, 접근된 데이터는 사용 즉시 데이터베이스에서 조회되어야 한다." 이런 설계를 기반으로 아키텍처는 "중앙 집중 관리되는 데이터베이스가 고객과 관련된 데이터를 관리한다."라는 아키텍처 결정사항을 정의하고, 중앙 집중식 데이터베이스 아키텍처 스타일(Shared Data Style)을 이용하여 아키텍처가 설계될 것이다.

- 아키텍처 모델 및 명세

"아키텍처 모델 및 명세"에서는 현재 뷰를 구성하는 모델들을 표현한다. 아키텍처 모델은 현재 뷰에서 보여지는 아키텍처를 추상적으로 또는 간단하게 표현하는 것이다. 아키텍처 모델을 이용함으로써, 현재 설계하고 있는 아키텍처를 더 쉽게 이해할 수 있으며 관련된 이해당사자와 의사소통하는데 도움이 된다.

- 적용 가능한 시나리오(선택적)

  해당 뷰 모델에서 반드시 나타내어야 하는 중요한 시나리오를 기술하는 장으로, 이 내용은 부록에 기술될 수도 있다. 만약 부록이 해당 뷰에 중요한 시나리오를 기술할 경우에는 "부록의 몇 장 몇 절을 참조하시오."라는 문구로 해당 문서를 참조하게 해야 한다.

  시나리오 작성은 선택적이지만, 작성된 시나리오는 설계된 아키텍처를 평가하거나 전체 시스템을 테스팅할 때 중요한 입력물이 될 수 있다. 앞서 2장에서 기술하였듯이, 설계된 아키텍처를 평가하는데 시나리오가 많이 사용되고 있다. 그러므로, 아키텍처 명세서를 작성할 때 시나리오를 기술하는 것은 노력과 시간이 많이 든다는 단점은 있지만, 아키텍처 평가하는데 중요한 자료가 된다.

  시나리오 작성 시에 다음과 같은 항목을 이용하여 기술한다.

  · 초기 시스템 상태과 시스템 주변 환경
  · 외부 자극(stimulus), 즉, 시스템의 기능 수행을 유발시키는 외부 이벤트
  · 요구되는 시스템 행위와 실제 수행된 시스템 행위

- 개발 시 주의사항(선택적)

o **"5장. 부록" 작성 지침**

아키텍처 명세서에 요구사항부터 아키텍처 설계까지 상세한 내용을 모두 다 포함한다면, 아키텍처 명세서의 분량은 방대해질 것이다. 그러므로, 아키텍처 명세서에 기술되는 1장부터 4장까지의 내용은 가능하면 간단하게 기술하고, 다른 문서를 참조하는 것이 좋다. 그리고, 만약 다른 문서에 기술하지 않았다면 부록을 이용하는 것도 다른 방법이 될 수 있을 것이다. 이는 아키텍처 명세서를 이해하게 쉽게 하면서 아키텍처에 대한 상세 내용도 유지할 수 있는 방빕이 된다. 부록에는 기관마다 다른 내용이 포함될 수 있지만, 다음과 같은 항목이 기술될 수 있다.

- 아키텍처 명세서에서 참조하는 다른 문서들에 대한 정보
- 용어 정리/ 약어 정리
- 시스템 범위, 기능적 요구사항 또는 비기능적 요구사항에 대한 상세한 내용
- 아키텍처 설계하면서 사용한 아키텍처 스타일, 디자인 패턴 등에 대한 상세한 설명

지금까지는 ADV_TDS의 계층과는 상관없이 모두 적용될 수 있는 전반적인 아키텍처 명세서의 구조에 대해 설명하였다. ADV_TDS의 1 단계(ADV_TDS.1 기본적인 설계)에서 2 단계(ADV_TDS.2 구조적인 설계)에는 비정형화된 표현인 텍스트 명세 기법을 이용하여 설계서를 작성하게 된다. 그러므로, 아키텍처 명세서의 1장부터 5장에 포함되어야 하는 모든 내용을 텍스트로 기술하면 되기 때문에 위에 기술된 명세 항목 외에는 별다른 지침 사항이 필요 없게 된다.

그러나, ADV_TDS 3 단계(ADV_TDS 3. 기본적인 모듈화 설계)에서 ADV_TDS 6단계(ADV_TDS 6. 정형화된 상위수준 설계 표현이 제공되는 완전한 준정형화된 모듈화 설계)까지에는 다양한 다이어그램으로 아키텍처 설계 모델을 표현할 수 있게 된다. 그러므로, 이제부터 ADV_TDS 3 단계에서 ADV_TDS 6 단계까지 적용될 수 있는 표현 다이어그램을 각 뷰별로 설명한다. 각 뷰에서 사용되는 다이어그램은 기본적으로 UML 다이어그램에 기반하고 있으며, 기관마다 각기 다른 다이어그램 표기법을 사용할 수 있다. 앞으로 기술되는 내용은 아키텍처 명세서의 "4장. 아키텍처 설계 모델"을 기술하는데 유용하게 사용될 수 있다.

o **기능 뷰(Functional View)**

기능 뷰에서는 시스템의 기능적인 능력(capability), 외부 인터페이스, 내부 인터페이스 등을 표현하며, 시스템의 기능적인 구조를 보여주는 다이어그램 등이 표현될 수 있다. UML 표준 다이어그램을 사용하건 기타 다이어그램을 사용하든지 상관없이 기능 뷰에서는 다음과 같은 요소를 포함해야 한다.

- 기능적인 요소 (Functional Element)

  기능적인 요소는 실시간에 운용되는 시스템에서 고유한 기능성(또는 책임, responsibility)을 가지는 단위로서, 외부 또는 내부의 다른 요소들과 상호작용하기 위해 잘 정의된 인터페이스를 가진다. 시스템을 나타내는 추상화 수준에 따라 달라지겠지만, 가장 하위 단계에서는 소스코드 모듈이 기능적인 요소가 될 수 있으며, 이외에 어플리케이션 패키지, 데이터 저장소, 또는 전체 시스템도 기능적인 요소가 될 수 있다. CC에서는 서브시스템과 모듈이 기능적인 요소가 될 수 있다.

  기능적인 요소를 기술할 경우에는 다음과 같은 항목이 표기되어야 한다.

  · 제공하는 기능의 특징을 잘 나타내는 요소의 이름

- 인터페이스 (Interface)

  인터페이스는 외부 또는 내부 다른 요소들과 접근하여 상호작용하는데 필요한 잘 정의된 메커니즘으로, 입력, 출력, 오퍼레이션의 시맨틱(semantic) 정보를 이용하여 기술될 수 있다. CC에서도 모듈 단위에서 인터페이스를 기술하는 요소로 이와 비슷한

항목들인 인터페이스가 제공하는 기능(오퍼레이션), 인터페이스의 반환값, 해당 인터페이스가 필요로 하는 다른 요소의 인터페이스, 인터페이스가 고유의 기능을 수행하는데 필요한 알고리즘 등이 나열된다.

인터페이스를 기술할 경우에는 다음과 같은 항목이 표기되어야 한다.

- 인터페이스에서 제공하는 오퍼레이션의 이름
- 입력 파라미터(input parameter)
- 반환값
- 그 외 제약사항 등

- 커넥터 (Connector)

커넥터는 한 기능적인 요소와 다른 기능적인 요소들이 어떻게 상호작용하는지를 연결시켜준다. 커넥터를 사용함으로써, 기능 요소들이 어떻게 상호작용하는지 나타내어주며, 호출될 오퍼레이션과는 상관없이 고려되어야 하는 상호작용 정보를 알려준다. 일반적으로 기능적인 요소 간의 연관관계(association)을 이용하여 표현할 수 있다.

- 외부 개체 (External Entities)

외부 객체는 다른 시스템, 소프트웨어 프로그램, 하드웨어 장치, 또는 개발되는 시스템과 상호작용하는 다른 개체들을 모두 포함한다. 외부 개체는 시스템의 컨텍스트를 정의한 정보에서 유도될 수 있다.

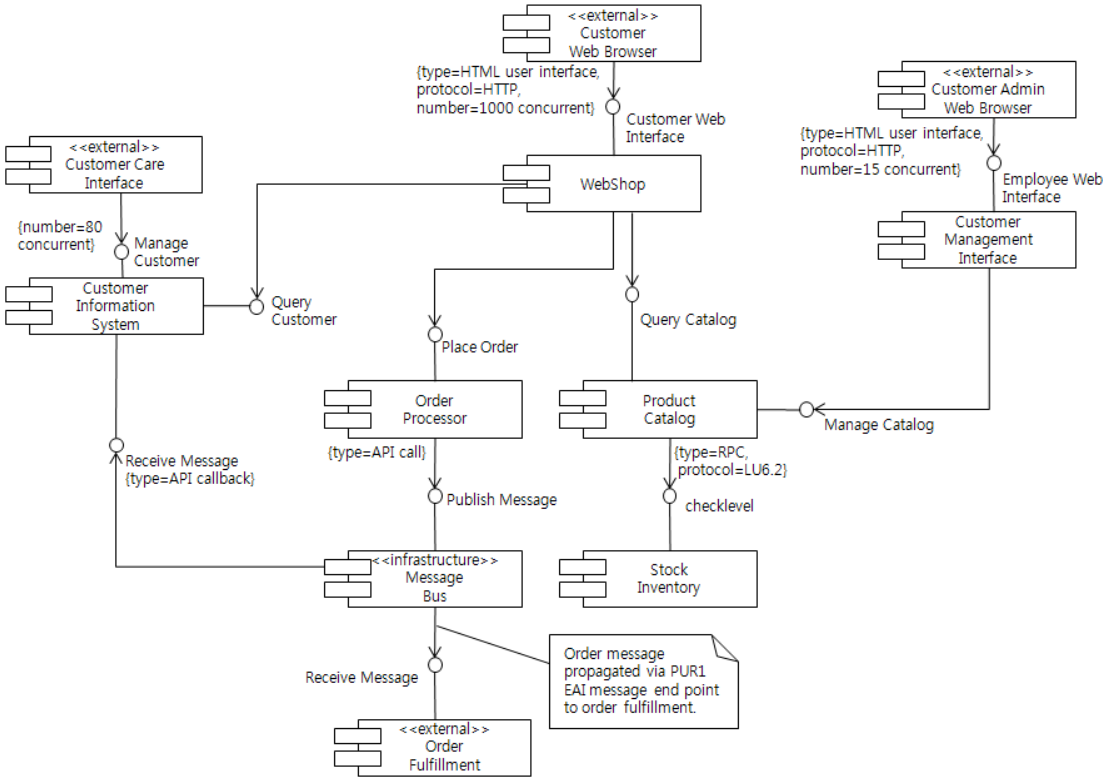
기능 뷰에 표현되는 다이어그램에는 프로세스와 쓰레드 등 코드가 어떻게 패키지화되고 실행되는지를 나타내는 개체보다는 기능 요소(컴포넌트 또는 클래스, CC에서는 서브시스템 또는 모듈)이 어떻게 구성되어 있는지를 표현해야 한다.

기능 뷰를 위한 다이어그램은 다양한 형태로 그려질 수 있다. UML 표준을 준수하는 경우에는 컴포넌트 다이어그램, 클래스 다이어그램 등이 사용될 수 있으며, 이 외에 단지 간단하게 박스와 선을 이용하여 다이어그램(Boxes-and-Line Diagram)을 표현할 수 있다.

- 컴포넌트 다이어그램 (Component Diagram)

서브시스템 단위로 보안 아키텍처를 설계하며, 준정형화된 표현을 사용하여 설계서를 작성 할 때, 컴포넌트 다이어그램이 기능 뷰에 잘 적용될 수 있다. 즉, ADV_TDS 3단계에서 기능 뷰를 나타내기 위해 컴포넌트 다이어그램을 유용하게 사용할 수 있다. 컴포넌트 다이어그램은 시스템 구성요소들을 모델링하는데 사용되고, 컴포넌트 단위로 시스템 구조를 표현하는데 사용한다. 그림 70은 기능 뷰를 나타내기 위해 컴포넌트 다이어그램을 사용한 예제를 보여준다. 시스템의 기능적인 요소와 외부 개체

는 UML의 컴포넌트 아이콘을 사용하여 나타내었고, 외부 개체인 경우에는 《external》 스테레오 타입을 이용하여 표현하였다. 《infrastructure》 스테레오 타입을 이용한 컴포넌트 아이콘은 시스템의 하부 구조(infrastructure) 요소를 나타낸다. 그림에서 롤리팝(lollypop) 아이콘을 이용해서 나타낸 것은 컴포넌트의 인터페이스를 나타낸다.



(그림 70) 기능 뷰를 나타내는 컴포넌트 다이어그램의 예제

- 패키지 다이어그램 (Package Diagram)

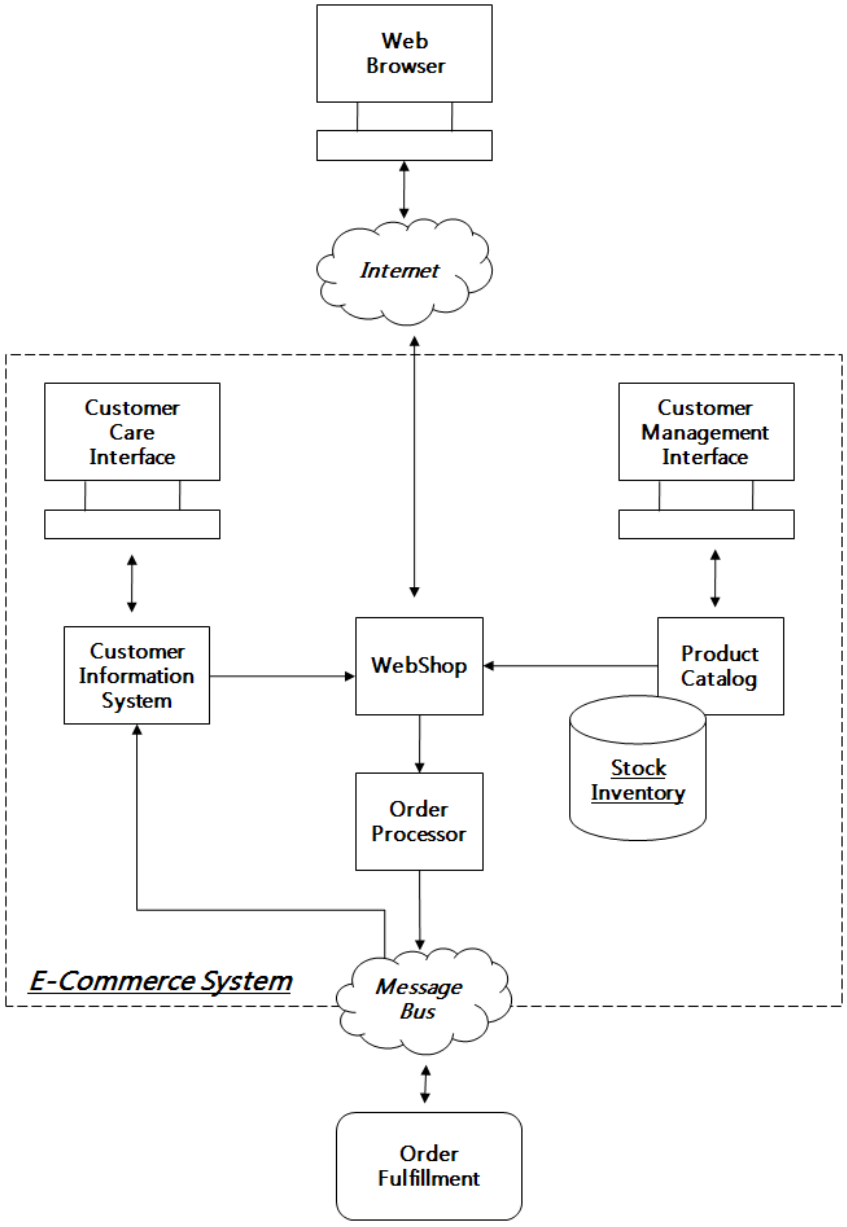
  컴포넌트 다이어그램 외에, 서브시스템 단위로 보안 아키텍처를 설계하며, 준정형화된 표현을 사용하여 설계서를 작성 할 때, 패키지 다이어그램 역시 기능 뷰에 잘 적용될 수 있다.

- 클래스 다이어그램 (Class Diagram)

  모듈 단위로 보안 아키텍처를 설계하며, 준정형화된 표현을 사용하여 설계서를 작성할 때, 클래스 다이어그램이 기능 뷰에 잘 적용될 수 있다. 즉, ADV_TDS 4단계에서 6단계까지의 기능 뷰를 나타내기 위해 클래스 다이어그램을 유용하게 사용할 수 있다. 클래스 다이어그램은 시스템의 구조를 보여주기 위해, 클래스와 클래스 간의 다양한 관계를 이용한다. 클래스는 이름, 속성, 오퍼레이션으로 구성되며, 클래스 간의 관계는 연관 관계(association), 상속 관계(generalization), 포함 관계(aggregation 또는

composition) 이 있다.

- 박스와 선을 이용한 다이어그램(Boxes-and-Line Diagram)
  위와 같이 UML 다이어그램을 이용하지 않더라도, 시스템의 기능적인 구조를 박스와 선을 이용하여 간단하게 표현할 수 있다. 이런 다이어그램은 시스템의 기능적인 요소와 각 요소의 인터페이스, 컨넥터의 역할을 하는 기능적 요소들이 어떻게 상호작용하는지를 나타내는 링크에 관련된 표기법을 포함한다. 단지 박스와 선을 이용하여 그린 다이어그램은 기술적인 지식이 없는 이해당사자도 쉽게 이해할 수 있다는 장점을 지니고 있다. 그림 71은 박스와 선을 이용한 다이어그램의 예제를 보여준다. 박스 표기법은 기능 뷰의 기능적인 요소들을 나타내며, 선은 기능적인 요소 간에 연관 관계가 있고 커넥터를 사용함을 나타낸다.



(그림 71) 박스와 선을 이용한 다이어그램의 예제

o 정보 뷰 (Information View)

정보 뷰에서는 아키텍처가 관리하고 제어하는 데이터/정보를 표현하며, 주로 정보 구조(information structure), 정보 흐름(information flow) 등을 나타낸다. 정보 뷰에서는 데이터와 데이터 간의 연관 관계를 나타내는 데이터 구조 모델(Data Structure Model)과 데이터가 실시간에 어떻게 교환하는지를 보여주는 정보 흐름 모델(Information Model) 관점에서 표현할 수 있다. 기능 뷰와는 달리, 정보 뷰에서는 ADV_TDS의 단계마다 사용될 수 있는 다이어그램에 차이가 있는게 아니라, 기술되는 데이터의 상세도 레벨에 차이만 있을 수 있다. 즉, 서브시스템 레벨에서 식별될 수 있는 데이터와 모듈 레벨에서 식별될 수 있는 데이터의 상세도 차이만 있을 뿐이지 사용되는 다이어그램을 비슷할 것이다.

- 데이터 구조 모델 (Data Structure Model)

  시스템 또는 소프트웨어 아키텍처에서 사용하는 데이터의 정적인 구조를 표현하는데 사용되며, 중요한 데이터 요소와 데이터 요소들 간의 관계를 나타낸다. 데이터 구조를 표현하기 위해서는 ER 다이어그램(Entity-Relationship Diagram)와 UML 다이어그램 중의 하나인 클래스 다이어그램이 사용될 수 있다.

  · ER 다이어그램

  ER 다이어그램은 데이터와 데이터 간의 관계를 표현해주는 대표적인 다이어그램이다. 관심 대상이 되는 데이터는 개체(Entity)로 나타내며, 개체는 속성(attribute)를 가진다. 그리고, 개체 간에는 관계(Relationship)가 있으며, 각 관계에는 한 개체의 인스턴스가 다른 개체의 몇 개의 인스턴스와 관련이 있는지를 나타내는 cardinatliry 정보를 가진다.

  · 클래스 다이어그램

  UML 다이어그램 중 하나로, 기능 뷰에서 시스템의 구조를 보여주기 위해 사용될 수 있지만 데이터 구조도 표현하는데 사용될 수 있다. 설계될 데이터는 클래스로 나타내며, 클래스는 속성을 가짐으로써 구성 데이터 요소를 관리할 수 있다. 그리고 데이터 간의 관계는 연관 관계(association)으로 표현할 수 있다. 클래스 다이엉그램에는 시스템의 정적인 구조를 보여주는 것 외에 오퍼레이션을 이용해서 동적인 측면도 표현할 수 있지만, 정보 뷰에서는 포함시키지 않는다.

- 정보 흐름 모델

  정보 흐름 모델은 실시간에 시스템의 데이터 요소 간 또는 외부 시스템과 어떻게 정보를 교환하는지를 보여준다. 즉, 데이터 요소와 데이터 요소 간의 흐름을 식별하는 것이 주요 목적이 되며, 흐름을 통하여 한 데이터가 다른 데이터로 변환되는 것을 보여줄 수 있게 된다. 정보 흐름 모델은 데이터가 많이 필요로 하는 시스템(Data-intensive System)에 효과적으로 사용될 수 있다.

  정보 흐름 모델을 위해서는 데이트 흐름 다이어그램(Data Flow Diagram, DFD)가 효과적으로 사용될 수 있다.

o 동시 뷰 (Concurrency View)

동시 뷰에서는 시스템이 실시간에 기능을 어떻게 수행하는지를 보여주며, 이외에도 시스템의 상태 변화도 보여준다. 즉, 동시 뷰를 통해 시스템의 행위를 보여준다. 정보 뷰와 마찬가지로, ADV_TDS의 단계마다 사용될 수 있는 다이어그램에 차이가 있는게 아

니라, 아키텍처 기능 요소(서브시스템 또는 모듈 레벨)에 따라서 기술되는 모델의 상세도 레벨에 차이만 있다.
동시 뷰를 위해서는 시스템 수준의 동시 모델(Concurrency Model) 또는 UML 다이어그램으로는 상태 머신 모델(State Machine Diagram), 시퀀스 다이어그램(Sequence Diagram) 등이 사용될 수 있다.

- 동시 모델 (Concurrency Model)

  동시 모델은 기능적인 요소(Functional Element)가 실시간 실행에 참여하는 개체로 어떻게 매핑이 되는지를 보여준다. 동시 모델에는 다음과 같은 항목들을 포함한다.

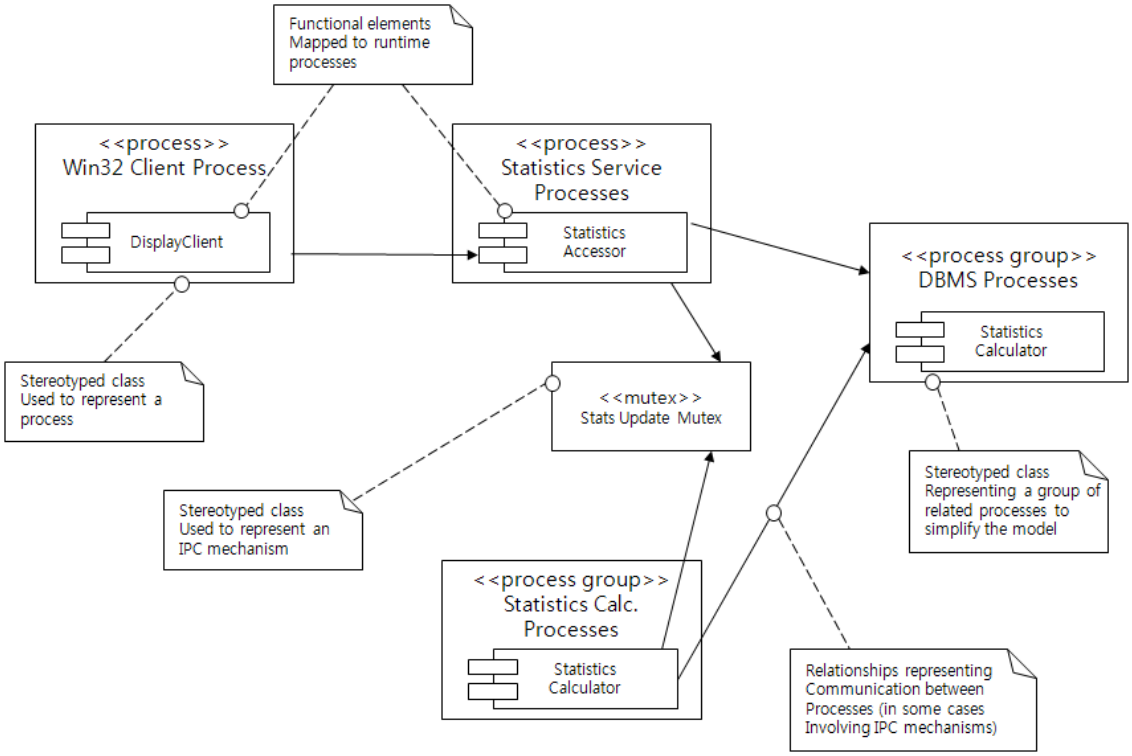
  · 프로세스 : 하나 이상의 쓰레드를 실행하기 위한 환경을 나타냄.
  시스템이 동시(병렬)에 여러 작업이 실행됨을 보여주는 기본 단위임.

  · 프로세스 그룹 : 연관 있는 프로세스들을 그룹화한 것.
  DBMS가 대표적인 프로세스 그룹의 예제임.

  · 쓰레드 : 한 운영 시스템 프로세스 내에서 개별적으로 수행될 수 있는 실행 단위.

  · 프로세스 간 상호작용

  UML 다이어그램에는 이를 효과적으로 나타내어주는 다이어그램은 없으므로, 그림 72와 같이 UML의 주요 표기법인 스테레오 타입을 인용하여 나타낼 수 있다. 그림 72에는 기능적인 요소를 위한 UML의 컴포넌트 아이콘을 실시간 프로세스로 매핑시켰고, 스테레오 타입을 이용하여 프로세스(《process》)와 프로세스 그룹(《process group》)을 표기하였다. 그리고 프로세스 간의 상호작용은 화살표를 이용하여 나타내었다.

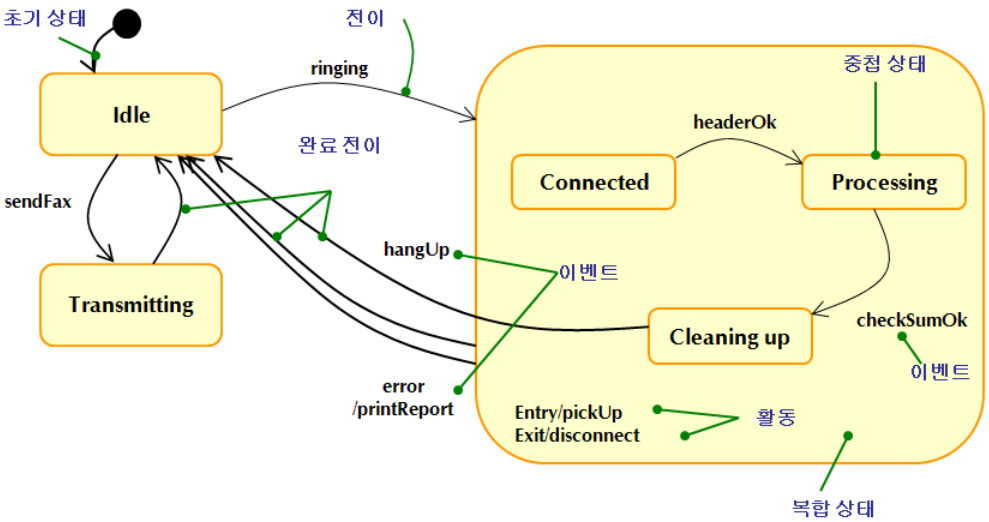


(그림 72) UML 표기법을 이용해서 기술한 동시 모델

- 상태 머신 다이어그램

상태 머신이 실시간에 어떻게 상태의 변화가 있는지를 모델링하며, 상태 머신은 는 객체가 조건에 따라 상태가 변하는 과정을 정의한다. 즉, 상태 머신 다이어그램은 시스템의 요소가 실시간에 보일 수 있는 상태들과 각 상태들이 전이 과정을 보여준다. 상태 머신 다이어그램은 다음과 같은 항목들을 포함한다.

- 상태 (State) : 실시간에 시스템이 가질 수 있는 식별가능한 기능적 요소의 상황임.
  예) 대기 상태(Waiting), 휴지 상태(Idle), 실행 중(Performing) 등
- 전이 (Transition)
  이벤트가 발생할 때 시스템의 기능적 요소가 한 상태에서 다음 상태로 이동하는 것을 나타내는 두 상태 간의 관계임.
- 이벤트 (Event)
  시스템에서 발생할 수 있는 중요하거나 관심이 될 수 있는 사건을 나타내며, 시스템 간의 전이를 유발시킨다.
- 활동 (Action)
  상태 전이와 연관될 수 있는 작업 처리의 단위로서, 이벤트는 시스템의 상태 전이를 유발시키지만, 활동은 시스템 전이의 한 부분으로서 실행된다. 그림 73은 상태 머신 다이어그램의 예제를 보여준다.



(그림 73) 상태 머신 다이어그램의 예제

- 시퀀스 다이어그램

시퀀스 다이어그램은 외부 사용자 또는 액터가 발생시키는 이벤트, 메시지 호출의 순서 등을 이용하여, 기능적인 요소들이 실시간에 어떤 메시지를 주고 받는지를 보여준다. 아키텍처 레벨에서는 상세한 수준의 기능적인 요소들이 식별되기 힘들지만, 서브시스템 단계에서 모듈 단계로 설계 수준이 상세화 됨으로써 기능성 단위가 작은 기능적인 요소들 간의 메시지 교환 관계를 보여줄 수 있다.

o 배치 뷰 (Deployment View)

배치 뷰에서는 시스템의 하드웨어, 소프트웨어 등의 다양한 요소들이 어떻게 배치되어 있는지를 보여주는 기술하다. 그러므로, 이 뷰에서는 요구되는 하드웨어 종류 및 각 하드웨어의 사양, 제 3자가 개발한 소프트웨어(third-party software), 네트워크 토폴로지(topology), 필요한 네트워크 사양 등의 내용들이 기술된다. 기능 뷰를 제외한 다른 뷰와 마찬가지로 배치 뷰 역시 ADV_TDS의 계층에 상관없이 사용가능한 다이어그램을 동일하며, 다만 다이어그램에 기술되는 요소들의 상세도 레벨에만 차이가 있을 뿐이다. 필요한 하드웨어 노드들을 표현하는 실시간 플랫폼 모델(Runtime Platform Model)와 네트워크 토폴로지를 나타내는 네트워크 모델(Network Model) 2가지 관점에서 배치 뷰를 표현할 수 있다.

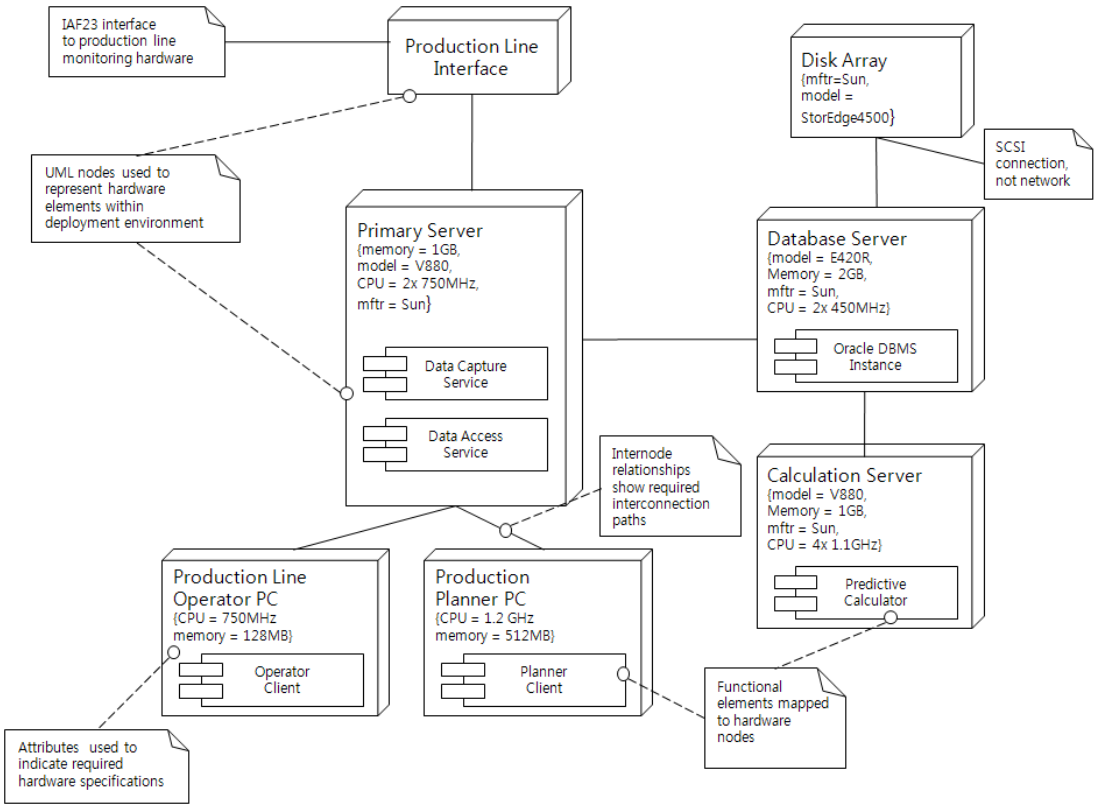
- 실시간 플랫폼 모델 (Runtime Platform Model)

실시간 플랫폼 모델은 배치 뷰의 핵심으로, 요구되는 하드웨어 노드들의 집합과 각 노드들이 네트워크를 통해 어떻게 연결되어 있는지, 하드웨어 노드에 배포된 소프트웨어는 무엇인지를 보여준다. 실시간 플랫폼 모델을 나타내기 위해 사용되는 다이어그램은 다를 수 있지만, 다음과 같은 항목들이 다이어그램에 표현된다.

- 프로세싱 노드
  시스템에 포함되는 컴퓨터 각각이 하나의 프로세싱 노드에 해당한다.
- 클라이언트 노드
  시스템을 사용하는 클라이언트가 사용하는 하드웨어를 나타낸다.
- 저장소 하드웨어
- 네트워크 링크
  시스템에 속한 다양한 노드들이 어떻게 연결되어 있는지를 보여준다.
- 기타 하드웨어 컴포넌트
  네트워크 보안이나 사용자 인증 등 특별한 목적으로 위해 필요한 하드웨어를 나타낸다.

배치 뷰의 이런 정보를 표현하기 위해서 UML의 배치 다이어그램 (Deloyment Diagram)과 표준 표기법이 아닌 간단하게 박스와 선을 이용한 다이어그램 (Boxes-and-lines Diagrams)을 이용할 수 있다.

- 배치 다이어그램(Deployment Diagram)
  실행 시점의 시스템 구조를 모델링하기 위해 사용한 다이어그램으로, 하드웨어 구성과 소프트웨어 배치 구성을 표현한다. 즉, 배치 다이어그램에는 소프트웨어가 구동되는 컴퓨팅 노드와 컴퓨팅 노드 간의 연결 관계를 나타낸다.
  그림 74는 배치 다이어그램을 이용하여 플랫폼 실시간 모델을 나타내는 예제이다. 실행 환경에 배포되는 하드웨어 요소(예, Primary Server)는 UML 노드를 이용하여 표현하였고, 기능적인 요소(예, Predictive Calculator)들은 컴포넌트 아이콘을 이용하여 표기되었고, 하드웨어 사양은 {...} 표기법 안에 기술하였다. 그리고, 노드 간의 연결관계는 선을 이용하여 표현하였다.



(그림 74) 실시간 플랫폼 모델의 예제

· 박스와 선을 이용한 다이어그램(Boxes-and-lines Diagrams)

기능 뷰와 마찬가지로, 간단하게 박스와 선을 이용하여 하드웨어 또는 소프트웨어가 어떻게 배치되는지를 표현할 수 있다. 박스는 하드웨어 노드 또는 소프트웨어 노드 등을 표현하며, 선은 노드 간의 연결 관계를 나타낼 수 있다.

- 네트워크 모델 (Network Model)

네트워크 모델은 선택적으로 기술되는 것으로, 네트워크 토폴로지가 간단하다면 실시간 플랫폼 모델에 같이 표현할 수 있다. 즉, 네트워크 모델은 실시간 플랫폼 모델의 일부분이라고도 할 수 있다. 네트워크 모델은 시스템의 네트워크 구조가 복잡할 경우에 유용하게 사용될 수 있고, 노드들이 네트워크 상에서 어떻게 연결되어 있는지, 네트워크를 위한 하드웨어 장비는 무엇인지, 네트워크의 대역폭 정보 등을 기술한다. 네트워크 모델에 기술되어야 하는 항목은 다음과 같다.

· 프로세싱 노드

데이터를 전송하기 위해 네트워크를 사용하는 시스템 요소들을 나타낸다. 프로세싱 노드는 실시간 플랫폼 노드의 프로세싱 노드로부터 유도될 수 있다.

· 네트워크 노드

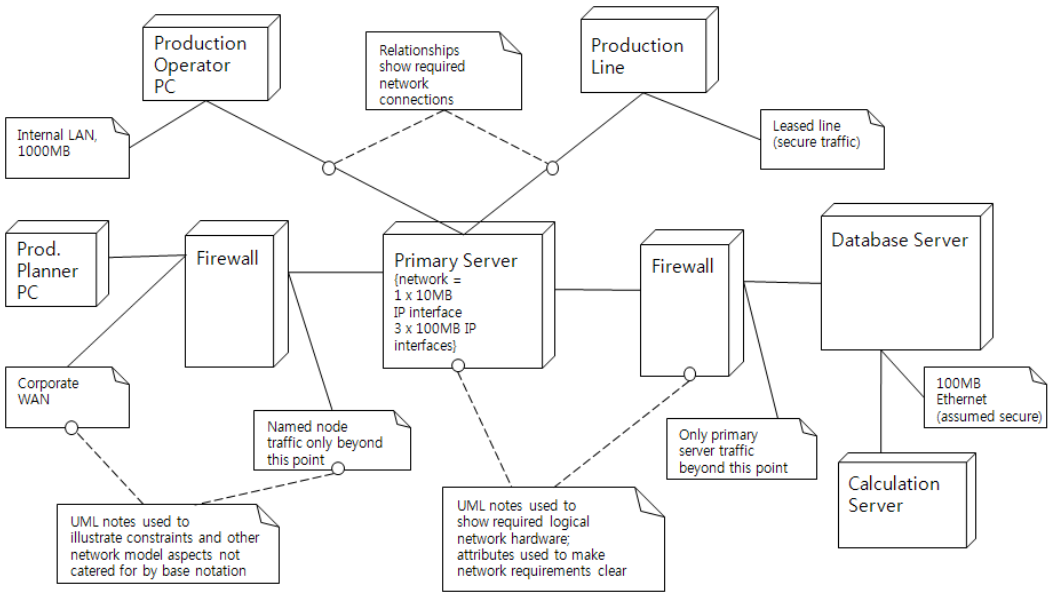
네트워크 서비스의 종류를 나타내며, 방화벽 보안(Firewall Security), 부하 균형(Load balancing), 암호화 등을 위한 서비스를 제공하는 노드를 나타낸다.

· 네트워크 연결 관계

네트워크 노드 또는 프로세싱 노드 간의 연결 관계를 나타내고, 대역폭 등 연결에 관련된 정보도 표현한다.

실시간 플랫폼 모델과 마찬가지로, UML의 배치 다이어그램 (Deloyment Diagram)과 표준 표기법이 아닌 간단하게 박스와 선을 이용한 다이어그램(Boxes-and-lines Diagrams)이 네트워크 모델을 표현하기 위해 사용될 수 있다.

그림 75는 배치 다이어그램을 이용하여 네트워크 모델을 나타낸 예제를 보여준다.



(그림 75) 네트워크 모델의 예제

위에서는 주로 ARC_TDS의 3단계에서부터 6단계까지 준정형화된 표현을 위해 사용될 있는 다이어그램에 대해 설명하였다. 그러나, ADV_TDS 6 단계인 ADV_TDS 6. 정형화된 상위수준 설계 표현이 제공되는 완전한 준정형화된 모듈화 설계은 정형화된 표현이 추가된다. 정형화된 표현은 앞서 말하였듯이, 수학적인 표기법을 이용하여 설계를 명세하는 것이다. 사용될 수 있는 명세 기법으로는 아키텍처 명세 언어(Architecture Description Language, ADL), 객체 제약 언어(Object Constraint Language, OCL), pseudo-code 등이 있다.

o 아키텍처 명세 언어 (ADL)

ADL은 시스템을 구현하기 이전에 시스템 아키텍처를 정의하고 설계하는데 사용된다. 시스템의 컴포넌트와 커넥터를 식별함으로써, ADL은 시스템 아키텍처 자체보다 더 상세한 내용을 포함한다. ADL 은 다음과 같은 항목들을 기술한다. 대표적인 ADL로는 ACME [9], Aesop [10], UniCon [11], Wright[12] 등이 있다.

- 컴포넌트 행위 명세 : 컴포넌트의 기능적인 측면과 비기능적인 측면을 모두 기술할 수 있다. 그리고, ADL 종류에 따라서, 시간 제약 사항, 컴포넌트 입력 자료, 출력 자

료, 데이터 등도 기술할 수 있다.

- 컴포넌트 프로토콜 명세 : Wright와 같은 일부 ADL은 컴포넌트의 상호작용 프로토콜을 명세할 수 있다.
- 커넥터 명세 : ADL을 통해서 커넥터의 속성, 커넥터가 컴포넌트간의 상호작용을 어떻게 지원하는지 등을 명세할 수 있다.

```
Component Simple is
      type in_type is ...
      type out_type is ...
      op f : in_type -> out_type
      op valid_input? : in_type -> Boolean
      port input_port : in_type
      port output_port : out_type
      axiom f-specification is
        (behavioral specification for the operation f)
      end axiom
      axiom valid_input?-specification is
        (behavioral specification for the operation
        valid_input?)
      end axiom
      interface is input_port!(x) ->
        ((output_port!f(x) -> Skip)
        < valid_input?(x) >
        (output_port!(Invalid_Data) -> Skip))
end Simple
```

(그림 76) ADL의 예제

참고 문헌

[1] Rozanski, N. and Woods, E., *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, Addison Wesley Professional, 2005.

[2] Bass, L., et al., *Software Architecture in Practice*, Addison-Wesley, 2003.

[3] Clements, P., et al., *Documenting Software Architectures Views and Beyond*, Addison-Wesley, 2003.

[4] Bushmann, F., *Pattern-Oriented Software Architecture: A System of Patterns, Volume 1*, Wiley, 1996.

[5] *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1, Revision 1*, September 2006.

[6] Kephart, O. and Chess, M., "The Vision of Autonomic Computing", *IEEE Computer*, IEEE Computer Society Press, Vol. 36, No. 1, pp. 41-50, January 2003.

[7] IBM Research Center, "An architectural blueprint for autonomic computing, Fourth Edition," Autonomic Computing White Paper, IBM, June 2006.

[8] IEEE Computer Society, *Recommended Practice for Architectural Description*, *IEEE P1471-2000*, Oct. 9, 2000.

[9] D. Garlan, R. Monroe, and D. Wile, "ACME: An Architecture Description Interchange Language," *In Proceedings of CASCON 1997*, November, 1997.

[10] D. Garlan, R. Allen, and J. Ockerbloom, "Exploring Style in Architectural Design Environments," In *Proceedings of SIGSOFT 1994: Foundations of Software Engineering*, pp. 175-188, November, 1997.

[11] M. Shaw, R. Deline, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them," *IEEE Transaction on Software Engineering*, vol. 21, no. 4, pp. 314-35, 1995.

[12] R. Allen, "A Formal Approach to Software Architecture," Ph. D. Thesis, Carnegie Mellon University, *CMU Technical Report CMU-CS-97-144*, 1997.

소프트웨어 아키텍처 기반 설계 모델 및 명세기법 개발

2007년 12월 인쇄
2007년 12월 발행

- 발행인: 황 중 여
- 발행처: 한국정보보호진흥원
  서울시 송파구 가락본동 78번지
  IT벤처타워(서관)
  Tel: (02) 4055-114
- 인쇄처: 한울 인쇄사
  Tel: (02) 2279-8494

<비매품>

1. 본 보고서는 정보통신부의 출연금 등으로 수행한 정보보호 강화사업의 결과입니다.
2. 본 보고서의 내용을 발표할 때에는 반드시 정보통신부 정보보호 강화사업의 결과임을 밝혀야 합니다.
3. 본 보고서의 판권은 한국정보보호진흥원이 소유하고 있으며, 당 진흥원의 허가 없이 무단 전재 및 복사를 금합니다.