KIPS Tr. Software and Data Eng.
Vol.6, No.8 pp.391~400 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.8.391

Practical Software Architecture Design Methods for Non-Conventional Quality Requirements

Hyun Jung La† · Soo Dong Kim††

ABSTRACT

Software architecture plays a key role in satisfying non-functional requirement (NFR), i.e. quality requirements and constraints. Architecture design methods and tactics for conventional NFR are largely available in literatures. However, the methods for the target system-specific non-conventional NFRs are not readily available; rather architects should invent the design methods from their experiences and intuitions. Hence, the hardship to design architectures for non-conventional NFRs is quite high. In this paper, we provide a systematic architecture design methodology for non-conventional NFRs. We provide a five-step process, and detailed instructions for the steps. In the process, we treat the traceability among artifacts and seamlessness as essential values for supporting effective architecture design. We apply the methodology on designing architectures for a platform software system. We believe that the proposed methodology can be effectively utilized in designing high quality architectures for non-conventional NFRs.

Keywords : Software Architecture, Non-Functional Requirement, Architectural Tactic, Design Process, Traceability-Based Evaluation

비전형적인 품질 요구사항을 고려한 실용적 소프트웨어 아키텍처 설계 기법

라 현 정† · 김 수 동††

요 약

소프트웨어 아키텍처는 비기능적 요구사항(Non-Functional Requirement, NFR), 즉 품질 요구사항과 제약사항을 만족시키는데 중요한 역할을 한다. 현재까지 진행된 대부분의 연구는 전형적인 NFR을 위한 아키텍처 설계 방법과 설계 택틱에 국한된다. 그러나, 목표 시스템에 특화된 비전형적인 NFR을 위한 설계 방법에 대한 연구는 많이 진행되고 있지 않고, 소프트웨어 아키텍트가 보유한 지식과 경험에 의해 비전형적인 NFR을 만족시킬 수 있는 효과적인 방법과 택틱을 유도하고 이를 기반으로 아키텍처를 설계한다. 그러므로, 비전형적인 NFR을 고려하여 아키텍처를 설계하는 효과적인 방법 및 택틱을 고안하는 것이 어렵다. 본 논문에서는 비전형적인 NFR을 만족시키는 소프트웨어 아키텍처를 설계하는 효과적이며 체계적인 아키텍처 설계 방법론을 제안한다. 이 방법론은 전형적인 NFR을 고려한 아키텍처 설계에도 적용될 수 있다. 제안된 방법론은 5개의 스텝으로 구성된 프로세스, 각 스텝에 대한 상세 활동 지침을 포함한다. 그리고, 제안된 프로세스가 잘 설계되었음을 보이기 위해, 산출물 간의 추적성 관계를 확인한다. 마지막으로, 제안된 방법론의 효율성과 실용성을 평가하기 위해 사례 연구를 수행한 결과를 제시한다.

키워드 : 소프트웨어 아키텍처, 비기능적 요구사항, 아키텍처 택틱, 설계 프로세스, 추적성 기반 평가

1. 서 론

소프트웨어 아키텍처는 기능적 요구사항뿐 아니라 비기능적 요구사항을 만족시키는데 중요한 역할을 한다[1]. 아키텍처 설계 프로세스는 일반적으로 아키텍처 스타일을 이용한 전반적인 구조 설계, 뷰포인트를 적용한 아키텍처 뷰 설계, 비기능적 요구사항(Non-Functional Requirement, NFR)을 만족시키는 아키텍처 택틱(Tactic)을 이용한 아키텍처 뷰 정제 활동을 포함한다.

NFR은 신뢰성, 보안, 효율성, 수정가능성과 같은 전형적인 NFR과 목표 시스템에 맞게 특화된 비전형적인 NFR로 분류될 수 있다. '의료진의 진단 결과와 소프트웨어에 의해 수행된 진단 결과 간의 차이를 최소화해야 한다'는 비전형적인 NFR의 한 예이며, 이는 IBM Watson Medical과 같은

※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2015R1A2A2A01004078).
† 종신회원 : ㈜스마티랩 연구소장
†† 종신회원 : 숭실대학교 소프트웨어학부 교수
Manuscript Received : February 3, 2017
Accepted : March 3, 2017
* Corresponding Author : Soo Dong Kim(sdkim777@gmail.com)

머신 러닝 기반의 의료 진단 시스템에서 요구된다.

NFR을 고려한 아키텍처 설계 기법은 소프트웨어 아키텍처 설계 연구 분야에서 기술적 난이도가 높으며, 앞으로도 많은 연구가 필요한 주제 중 하나이다[1]. 소프트웨어 아키텍처 연구를 많이 하는 Software Engineering Institute (SEI) 연구소는 품질 기반 설계 기법(Attribute-Driven Design, ADD)를 제안하며[2, 3] 특정 NFR에 대한 ADD 적용 사례를 통해 품질 요구사항 기반의 소프트웨어 아키텍처 설계법을 제시한다[4, 5]. Rozanski의 책에는 10가지 품질 속성을 적용하는 방법을 다루고 있다[6]. 이렇듯, 현재까지 진행된 대부부의 소프트웨어 아키텍처 연구는 특정 NFR을 위한 아키텍처 설계 방법에 국한된다. 그러나, 목표 시스템에 특화된 비전형적인 NFR을 위한 설계 방법에 대한 연구는 많이 진행되고 있지 않고, 소프트웨어 아키텍트가 보유한 지식과 경험에 의해 비전형적인 NFR을 만족시킬 수 있는 효과적인 방법과 택틱을 유도하고 이를 기반으로 아키텍처를 설계한다. 그러므로, 비전형적인 NFR을 고려하여 아키텍처를 설계하는 효과적인 방법 및 택틱을 고안하는 것이 어렵다.

그렇기 때문에 비전형적인 NFR을 위해 아키텍처를 설계하는 효과적인 방법론이 필요하다. 본 논문에서는 비전형적인 NFR을 만족시키는 소프트웨어 아키텍처를 설계하는 효과적이며 체계적인 아키텍처 설계 방법론을 제안한다. 이 방법론은 전형적인 NFR을 고려한 아키텍처 설계에도 적용될 수 있다. 제안된 방법론은 5개의 스텝으로 구성된 프로세스, 각 스텝에 대한 상세 활동 지침을 포함한다. 그리고, 제안된 프로세스가 잘 설계되었음을 보이기 위해, 각 스텝의 산출물의 구성 요소를 나열하고, 산출물 간의 추적성(Traceability) 관계를 확인을 통한 검증 방법을 포함한다. 마지막으로, 제안된 방법론의 효율성과 실용성을 평가하기 위해 사례 연구 수행 결과를 보여준다.

본 논문은 다음과 같이 구성된다. 2장에서는 NFR을 고려한 소프트웨어 아키텍처 설계 기법에 대한 관련 연구를 분석한다. 3장에서는 소프트웨어 아키텍처의 핵심 입력물과 산출물을 정리한다. 4장에서는 비전형적인 NFR을 고려한 실용적인 아키텍처 설계 프로세스가 제시되고, 프로세스를 구성하는 각 스텝을 수행하는데 필요한 상세 지침을 정의한다. 각 스텝은 출력물의 구성요소를 반정형(Semi-formalism) 형식으로 기술하며, 출력물의 산출물을 제시한다. 그리고, 주어진 NFR이 충분히 반영되었는지를 효과적으로 평가하기 위해 산출물 간 추적성 관계를 이용한 NFR 기반 아키텍처 설계 명세의 평가 기법을 제안한다. 마지막으로, 5장에서 비전형적인 NFR을 만족해야 하는 소프트웨어 플랫폼 아키텍처 설계에 제안된 방법론을 적용하고, 그 결과를 포함한다.

제시된 방법론을 준수하여 아키텍처를 설계하면 비전형적인 NFR을 만족하는 고품질의 아키텍처를 보다 효과적으로 설계할 수 있을 것이라고 예상된다. 그리고, 추적성 관계를 활용하여 NFR을 충분히 반영하여 아키텍처를 설계하는데 도움이 될 것이라고 사료된다.

2. 관련 연구

소프트웨어 아키텍처의 주요 연구 분야 중 하나인 품질 요구사항을 반영한 아키텍처 설계에 대한 여러 연구가 진행되어 왔다.

SEI 연구소가 제안한 ADD는 품질 요구사항을 만족시키기 위하여 목표 시스템을 세분화하면서 반복적으로 아키텍처 택틱 또는 패턴을 적용하는 아키텍처 설계 프로세스로서, 총 7개의 스텝으로 구성된다[2, 3]. 세분화할 시스템 컴포넌트를 식별하고, 이 컴포넌트 설계에 필요한 아키텍처 드라이버를 도출하며, 아키텍처 택틱 또는 패턴을 적용하여 시스템 컴포넌트를 상세히 설계하는 과정을 거친다. 그리고, 특정 NFR에 대한 ADD 적용 사례를 통해 품질 요구사항 기반의 소프트웨어 아키텍처 설계 방법을 제안하였다[4, 5]. ADD는 품질 요구사항을 주요하게 다루는 프로세스이지만, 각 스텝을 수행하는데 필요한 산출물 템플릿 등 상세한 지침이 보완될 필요가 있다.

Kim의 연구는 아키텍처 설계에 고려된 아키텍처 택틱 간의 관계를 효과적으로 명세하기 위한 표기법에 대한 것이다[7]. 이 연구에서는 아키텍처 설계에 적용된 아키텍처 택틱 간의 관계를 휘처 모델로 표현하였고, 각 택틱의 시맨틱 정보는 역할 기반 메타모델링 언어로 기술하였다. 이 연구는 비기능적 요구사항을 설계하는 지침보다는 아키텍처 명세에 더 초점이 맞추어져 있다.

Tsadimas의 연구는 모델 기반 접근법을 채택하여 엔터프라이즈 정보 시스템(Enterprise Information System, EIS)의 아키텍처 설계에 NFR을 효과적으로 반영할 수 있는 방법에 대한 것이다[8]. 이들은 EIS 아키텍처 모델을 표현하기 위한 기능 뷰, 토폴로지 뷰, 네트워크 인프라스트럭처 뷰, NFR 뷰를 제안하였다. 특히 NFR 뷰는 NFR과 다른 3가지 뷰들 간의 상호 관계를 잘 표현할 수 있다. 그리고, 뷰들 간의 관계를 표현할 수 있는 SysML 프로파일을 정의하였다. 이 연구도 비기능적 요구사항이 아키텍처 설계에 잘 반영되었는지에 대한 명세에 더 초점이 맞추어져 있다.

Reza와 Grant의 연구는 주어진 비기능적 요구사항을 만족시킬 수 있는 최적의 아키텍처 스타일을 선정하는 기법에 대한 것이다[9]. 이 연구에서는 품질 요구사항, 디자인 택틱, 아키텍처 스타일 간의 상호 연관 관계를 트리 구조로 정의하여, 이 트리 구조의 연관 관계를 최적의 스타일 선정 가이드라인으로 활용할 수 있도록 하였다. 그러나, 이 연구는 잘 알려진 일반적인 NFR에만 제한적으로 적용된다.

Pedraza-Garcia의 연구에서는 보안 택틱을 아키텍처 설계에 적용하는 프로세스가 제안되었다[10]. 이 프로세스는 보안 요구사항과 보안에 민감한 자원 식별, 보안 정책 정의, 보안 택틱 식별, 보안 택틱을 아키텍처에 적용 및 명세 등의 순서로 진행된다. 이 연구 결과는 보안 요구사항에만 적용된다는 한계를 가지고 있다.

이 외에 소프트웨어 아키텍처에 관련된 저명한 서적인 Rozanski의 책에는 10가지 품질 속성을 적용하는 방법을 다

루고 있다[6]. 이 책에는 주요 품질 요구사항을 만족시키기 위한 택틱 및 메소드가 잘 정의되어 있지만, 책에서 다루어지고 있지 않은 품질 요구사항을 아키텍처 설계에 반영하는 지침은 부족하다.

현재까지 진행된 연구 결과를 요약하면, 대부분의 연구는 아키텍처 설계에 관련 택틱이 어떻게 반영되었는지를 명세하는 기법 또는 잘 알려진 NFR을 고려한 아키텍처 설계 기법에 초점이 맞추어져 있다. 즉, 일반적이지 않고, 특정 애플리케이션에만 국한된 비기능적 요구사항을 고려한 아키텍처 설계 기법에 대한 연구는 많이 진행되지 않고 있다. 그리고, NFR을 고려한 아키텍처 설계에 대한 실용적인 방법이 보완되어야 한다. 구체적으로, NFR이 아키텍처 설계 산출물에 어떻게 영향을 미치는지 등에 대한 상세한 가이드라인 및 산출물 등이 보완되어야 한다.

3. 기반 연구

소프트웨어 아키텍처는 목표 시스템을 개발하는 과정에서 비기능적 요구사항을 만족시키는 유일한 수단이다. 소프트웨어 아키텍처 설계 프로세스에 대한 표준은 없지만, 일반적으로 다음과 같은 활동으로 구성된다[6].

- 시나리오 작성을 통한 아키텍처 요구사항 분석
- 목표 시스템을 위한 아키텍처 스타일 선정 및 통합
- 아키텍처 설계 택틱 등 아키텍처 설계 기법 적용
- 최종 소프트웨어 아키텍처 평가

위의 과정을 거쳐 목표 시스템의 아키텍처에 대한 모든 설계 결정사항을 포함한 아키텍처 명세서를 작성한다. 소프트웨어 아키텍처 명세서에 관한 ISO/IEC 42010에 따르면[11], 아키텍처 명세서는 하나 이상의 뷰, 아키텍처 설계 근거(Rationale), 아키텍처 뷰포인트(Viewpoint)로 구성되며, 아키텍처 설계 결정사항을 보여주는 뷰는 하나 이상의 모델을 이용하여 표현된다.

본 논문에서 제시한 프로세스는 위에서 설명한 아키텍처 설계 전체 프로세스 중 세 번째 활동에 국한된다. 즉, 본 논문에서 제시할 프로세스를 적용하기 이전에 목표 시스템의 기능적 요구사항에 대한 이해를 기반으로 작성되며, 일반적으로 요구사항을 잘 해결할 수 있는 하나 이상의 아키텍처 스타일을 선정하고, 목표 시스템 개발 과정에 사용되는 뷰포인트(Viewpoint)들을 적용하여 기능적 뷰포인트, 정보 뷰포인트, 행위 뷰포인트, 배치 뷰포인트 등의 초기 버전을 작성한다. 초기 버전의 아키텍처 뷰 모델에는 비기능적 요구사항이 충분히 반영되어 설계되어 있지 않기 때문에, 본 논문에서는 NFR을 모두 만족시킬 수 있는 아키텍처 설계 방법에 국한된 프로세스를 제안한다. 그리고, 제시된 프로세스는 ISO/IEC 42010 표준에서 명시된 아키텍처 명세서를 준수한 산출물을 작성하도록 정의된다.

4. 비기능적 요구사항 반영을 위한 프로세스

NFR을 위한 소프트웨어 아키텍처를 설계하는데 기술적 어려움이 발생했을 때, 소프트웨어 아키텍트는 그들의 프로젝트 경험과 덜 체계적인 방법을 의존하게 되며[1], 결과적으로 주어진 NFR을 충분히 만족시키지 못하는 아키텍처를 설계하게 된다.

이러 어려움을 극복하기 위해, 본 논문에서는 비전형적인 NFR을 위한 아키텍처를 설계하는데 활용될 수 있는 체계적인 프로세스를 제안한다. 이 프로세스는 전형적인 NFR을 위한 아키텍처 설계에도 적용될 수 있다. 제안된 프로세스는 Fig. 1과 같이 비전형적인 NFR의 이해도를 높이기 위한 NFR 분석(Analyzing NFR), NFR을 위한 아키텍처 설계(Designing Architecture for NFR), 아키텍처 평가(Validating the Resulting Architecture)의 3가지 활동을 위한 5개의 상세 스텝으로 구성된다.

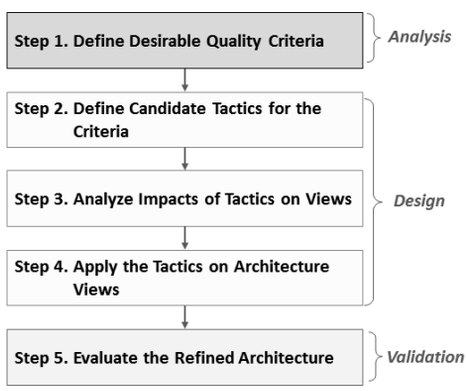

Fig. 1. Overall Process

스텝 1에서는 아키텍처 설계에 필요한 준비 작업을 하며, 스텝 2부터 스텝 4에서는 주어진 NFR을 고려한 실제 아키텍처를 설계한다. 그리고, 스텝 5에서 설계된 아키텍처가 주어진 NFR을 만족하고 있는지에 대해 평가한다.

4.1 스텝 1. 요구사항 만족 기준 식별

개요: 스텝 1에서는 비전형적인 NFR을 보다 효과적으로 이해하는데 도움이 되기 위해, NFR을 만족시키는데 고려되어야 하는 기준을 도출한다. 전형적인 NFR은 이미 잘 알려져 있는 요구사항이기 때문에, 이를 정확히 이해하는 것이 어렵지 않다. 그러나, 비전형적인 NFR은 목표 시스템에 특화된 요구사항이므로, 아키텍처 설계에 반영하기 이전에 이를 정확히 이해해야 한다.

입/출력물: 본 스텝의 입력물은 목표 시스템의 개발의뢰한 클라이언트로부터 받은 요구사항 명세서(Software Requirement Specification, SRS)이다. 요구사항 명세서는 기능적 요구사항과 비기능적 요구사항(NFR)으로 구성되며, 특히 NFR이 이 스텝을 수행하는데 필요한 가장 중요한 입력물이 되며, 다음과 같이 표현된다.

NFRSet = {NFR_i | i = 1, 2, …l, NFR_i는 NFR 한 항목이며, l은 목표 시스템에 고려되어야 하는 NFR 수임.}

이 단계의 출력물은 각 NFR을 상세하게 설명하는 요구사항 만족 기준이 되며, 다음과 같이 m 개의 요구사항 만족 기준으로 구성된다. 한 NFR_i로부터 복수 개의 만족 기준(C_j)이 유도되기 때문에, 요구사항 만족 기준의 총 개수인 m은 일반적으로 NFR의 총 수인 l과 같거나 작다.

CriteriaSet = {C_j | j = 1, 2, …, m, C_j는 특정 요구사항 만족 기준이며, m은 NFR을 만족시키는 기준의 수임.}

그리고, 각 C_j는 Table 1과 같이 만족 기준 ID, 만족 기준 이름, 만족 기준에 대한 설명으로 작성된다.

Table 1. Template for Desirable Criteria Identification Table

Criteria ID	Name	Description

첫 번째 열에는 평가에 활용하기 위한 만족 기준 식별자(ID)를 작성하고, 두 번째와 세 번째 열에는 각 만족 기준에 대한 이름과 이에 대한 상세 설명을 작성한다.

지침: 비전형적 요구사항 만족 기준을 효과적으로 도출하기 위해, 먼저 클라이언트가 정의한 NFR와 관련 있는 Fact 및 정책(Policy)을 식별한다. 여기서 근간이 되는 Fact라고 하는 것은 주어진 NFR과 관련되어 이미 알려져 있는 사실, 가정들을 의미하며, 정책은 해당 NFR을 만족시키는 프로세스에서 관찰되거나 사용될 수 있는 규칙, 규제, 법 등을 의미한다. 예를 들어, 인터페이스를 사용하는데 불일치가 발생해서는 안 된다는 인터페이스 호환성에 대한 NFR이 요구된다고 가정해보자. 일반적으로 인터페이스는 오퍼레이션 이름, 입력 매개변수, 반환 타입으로 정의된다. 이를 기반으로 인터페이스 사용에서 발생할 수 있는 비호환성 관련 문제는 오퍼레이션 이름 불일치, 입력 매개변수 불일치, 반환 타입 불일치로 분류될 수 있다. 그러므로, 인터페이스 호환성을 만족시키기 위해서는 1) 오퍼레이션 이름, 2) 입력 매개변수, 3) 반환 타입의 호환성이 유지되어야 한다는 Fact를 식별할 수 있다.

둘째, 식별된 Fact와 정책을 기반으로 주어진 NFR을 만족시키는데 고려되어야 하는 기준을 도출한다. 여기서 기준은 NFR을 만족시키는데 도움이 되는 기준 또는 피해야 하는 잠재적인 위험 기준도 될 수 있다. 그러므로, 요구사항 만족 기준에 대한 일반적인 명세는 “주어진 NFR과 관련된 품질 속성을 증가시켜야 한다” 또는 “주어진 NFR을 저해하는 위험 요소를 피한다”라는 형태로 기술된다. 예를 들어, 앞서 인터페이스 호환성 요구사항에 대한 만족 기준은 “오퍼레이션 이름 호환성을 만족시켜야 한다”, “오퍼레이션 입력 매개변수의 호환성을 유지해야 한다” 형태로 기술된다. 이 결과는 Table 1과 같은 요구사항 만족 기준 식별표를 이용하여 작성한다.

4.2 스텝 2. 후보 택틱 평가

개요: 스텝 2에서는 후보 아키텍처 택틱(Tactic)을 정의하고, 이 택틱들의 적용 가능성 및 장/단점 등을 분석하여 목표 시스템의 아키텍처 설계에 실제로 적용될 택틱들을 선정한다. NFR 만족 기준을 충족시킬 수 있는 택틱들을 선정하였다고 하더라도, 택틱을 적용함으로써 오버헤드 발생 또는 추가 자원 소모 등의 부작용이 발생할 수 있다. 그러므로, 각 택틱들을 분석하여 목표 시스템에 적용할 택틱 집합을 도출한다.

입/출력물: 본 스텝의 입력물은 이전 스텝의 산출물인 요구사항 만족 기준 목록이다. 그리고, 산출물은 각 요구사항 만족 기준을 해결하기 위해 선정된 아키텍처 설계 택틱 목록이다. 아키텍처 설계 택틱 목록은 다음과 같이 n 개의 아키텍처 설계 택틱으로 구성되며, n은 일반적으로 요구사항 만족 기준의 총 개수인 m과 같거나 크다.

TacticSet= {T_k | k = 1, 2, …n, n은 목표 시스템에 적용될 최종 택틱의 수}

최종적으로 선정된 아키텍처 설계 택틱은 Table 2와 같이 택틱 이름, 선정 결과, 택틱과 관련된 요구사항 만족 기준 ID, 선정 근거로 기술된다.

Table 2. Template for Tactic Evaluation Results

Tactic ID	Name	Decision (Y/N)	Criteria ID	Rationale
				[In terms of Applicable Situation]
				[In terms of Pros & Cons]

첫 번째 열과 두 번째 열에는 후보 택틱의 ID와 이름을 작성한다. 세 번째 열은 각 택틱을 목표 시스템에 적용할지 여부를 Y 또는 N으로 작성한다. 네 번째 열에는 각 택틱이 만족시킬 수 있는 만족 기준 ID를 작성한다. 다섯 번째 열은 세 번째 열에 작성한 적용 결과 결정 사항에 대한 근거를 작성하는 것이다. 이 근거는 1) 적용 가능성 일치 여부, 2) 택틱 적용 시의 장점 및 택틱 적용 시의 잠재적인 부작용 및 효과적인 해결 방법 여부를 고려하여 작성한다.

지침: 먼저, 이전 스텝에서 도출한 기준을 만족시키는데 활용될 수 있는 후보 아키텍처 택틱을 제안한다. 아키텍처 택틱은 일반적으로 특정 알고리즘, 디자인 패턴, 재사용 소프트웨어 자산 등을 포함하며, 소프트웨어 분석 및 설계에 활용될 수 있는 설계 기법, 가이드라인 등에 해당된다. 이런 택틱은 소프트웨어 설계 관련 문헌, 웹 사이트 등에서 찾아볼 수 있으며, 혹은 소프트웨어 아키텍트가 기존 여러 프로젝트 경험에 의해 얻은 설계 자산이 될 수 있다.

둘째, 후보 택틱들을 목표 시스템에 적용할 경우의 장/단점을 분석하며, 이들 간의 Trade-off 분석을 통해 목표 시스템에 적용할 최종 택틱을 선정한다. 이런 과정을 통해, 택틱을 적용할 경우의 단점을 최소화하고, 장점을 극대화하도록 한다. 후보 택틱을 평가한 결과는 Table 2와 같이 택틱 평가 결과표를 이용하여 작성한다. 후보 택틱의 적용 결과가 Y로 되

어 있는 경우에는 해당 택틱을 적용해야 하는 이유를 작성하며, 적용 가능한 상황이 일치하며 택틱 적용으로 인한 장점을 잘 활용할 수 있다는 형식으로 작성된다. 그리고, N로 되어 있는 경우에는 왜 해당 택틱을 목표 시스템 아키텍처 설계에 반영하지 않는지를 타당성 있게 작성하며, 대부분 택틱 적용으로 인한 부작용이 있는 경우에 선택되지 않는다.

이 스텝을 수행할 때 유의할 점은 NFR 만족 기준 별 최소 하나 이상의 택틱이 결정되어야 한다는 것이다. 적용 결과 평가에서 특정 기준을 위한 택틱이 하나도 없으면, 후보 택틱을 다시 선정해야 한다.

4.3 스텝 3. 택틱 별 뷰 변경 범위 분석

개요: 스텝 3에서는 스텝 2에서 선정한 택틱을 목표 시스템에 적용할 경우 초기 버전으로 작성한 아키텍처 뷰 모델에 미치는 변경 범위를 분석한다. 택틱은 초기 버전의 아키텍처 모델에 NFR을 적용하기 위해서 반영된다[6].

입/출력물: 본 스텝의 입력물은 이전 스텝의 산출물인 아키텍처 설계 택틱 목록이다. 그리고, 산출물은 각 택틱이 어떻게 아키텍처 뷰에 반영되는지를 기술한 뷰 변경 평가 결과이다. 선정된 n개의 택틱으로 정제되는 뷰는 다음과 같이 x개로 구성되며, 하나의 택틱 선정으로 복수 개의 뷰가 정제될 수 있다. 이 스텝의 수행 결과는 다음과 같이 뷰를 어떻게 정제하는 지에 대한 지침 집합이다.

ImpAnalysisSet = {(V_x, $Impact_y$)| x = 1, 2, …, υ, V_x는 택틱 적용으로 변경될 뷰이며, υ는 목표 시스템의 아키텍처를 표현하는 뷰의 개수, $Impact_y$는 각 뷰를 어떻게 수정해야 하는지에 대한 분석 결과}

여기서, *ImpAnalysisSet*은 V_x와 $Impact_y$ 쌍의 집합이 되며, Table 3과 같은 템플릿을 이용하여 작성된다.

Table 3. Template for Tactic Impact Analysis Table

View / Tactic ID	Functional View	Information View	Behavioral View	Deployment View	Other Views

첫 번째 열은 스텝 2에서 선정된 n개의 택틱들을 나열하고, 그 외의 나머지 열에는 택틱을 적용할 때 각 뷰들이 어떻게 변경되는지를 작성한다. 즉 *(V_x, $Impact_y$)*에 대한 내용를 작성한다. 본 템플릿은 아키텍처 설계에 보편적으로 많이 사용되는 기능 뷰, 정보 뷰, 행위 뷰, 배치 뷰에 각 택틱이 어떻게 적용되는지를 명세할 수 있으며, 목표 시스템에서 추가적인 뷰를 고려하고 있으면 별도의 열을 추가하여 작성하면 된다.

지침: 이 스텝에서는 선택된 택틱들을 효과적으로 적용하기 위해 각 택틱이 목표 시스템 아키텍처에 어떻게 반영되는지를 분석하며, Table 3과 같은 택틱 적용 변경 평가 표를 작성한다. 특히, 평가 표의 각 셀에는 각 뷰의 어떤 모델이 어떻게 변경되었는지를 기술하며, 초기 버전 뷰 모델의 변경에 대한 대표적인 명세는 다음과 같다.

- 기능 뷰의 ___ 모델에 특정 컴포넌트 추가 또는 삭제
- 기능 뷰의 ___ 모델에 컴포넌트 간 관계, 컴포넌트의 속성 및 오퍼레이션 등 수정
- 정보 뷰의 ___ 모델에 데이터 컴포넌트 추가 또는 삭제
- 정보 뷰의 ___ 모델에 데이터 컴포넌트 간 관계, 데이터 컴포넌트의 속성 수정
- 행위 뷰에 특정 알고리즘 반영한 새 다이어그램 작성
- 행위 뷰의 ___ 모델에 작성된 기존 알고리즘을 수정
- 배치 뷰의 ___ 모델에 노드 추가 및 삭제
- 배치 뷰의 ___ 모델에 노드 간 관계 수정
- 배치 뷰의 ___ 모델에 각 노드의 산출물 내용 수정

위 명세 예에서 보듯이, Table 3의 각 셀에는 V_x에 반영되는 택틱 적용 지침인 $Impact_y$가 기술되며, 이는 택틱이 적용되는 모델 이름과 어떻게 적용되는지에 대한 지침을 상세히 기술해야 한다. 여기서 모델 이름을 구체적으로 기술하는 이유는 한 뷰에 복수 개의 모델이 포함될 수 있기 때문이다. 이렇게 상세히 기술된 택틱 적용 지침은 다음 스텝에 아키텍처 정제 시에 활용된다.

4.4 스텝 4. 각 뷰에 택틱 적용

개요: 스텝 4에서는 스텝 3에서 분석한 택틱 적용 변경 범위 평가 결과를 이용하여 각 아키텍처 뷰를 수정한다. 이전 스텝에서 이미 각 뷰들이 어떻게 수정될 것인지를 상세히 분석하였기 때문에, 이 스텝에서는 이 내용을 이용하여 기존 아키텍처 뷰를 수정한다.

입/출력물: 본 스텝의 입력물은 이전 스텝의 산출물인 택틱 별 뷰 변경 평가 결과이다. 그리고, 산출물은 다음과 같이 소프트웨어 아키텍처 명세서에 포함되는 각 뷰 별 정제된 모델 집합이다.

ModelSet= {M_z | z = 1, 2, …w, M_z는 택틱 적용으로 변경된 모델이며, w는 그 모델의 수}

그리고, 택틱이 잘 적용되었는지 확인하는데 도움이 되기 위해, 정제된 부분에 대한 설명을 작성해야 한다.

지침: 소프트웨어 아키텍트는 택틱 하나씩 기존 아키텍처 뷰 모델에 반영하는 과정을 반복하여, 모든 택틱를 아키텍처에 잘 반영하도록 해야 한다.

각 택틱을 반영하면 각 다이어그램의 일부가 변경된다. 그리고, 이렇게 변경된 부분은 색상을 다른 도형을 사용하거나, 스테레오 타입을 이용하여 적용된 택틱의 식별자를 작성하는 것이 좋다. 이렇게 하면, 다음 단계에서 아키텍처 평가 시에 각 택틱이 올바르게 적용되었는지 확인하는 데에 도움이 된다.

Fig. 2는 택틱 적용으로 새로운 클래스가 추가된 경우를 보여준다. 이 때 새롭게 추가된 클래스인 SESS HW State는 다른 클래스와 다르게 노란색으로 하이라이트하여, 택틱 적용으로 변경된 부분을 표현하였다. 그리고, 이 다이어그램

의 명세 부분에 택틱 적용으로 하드웨어 상태를 기록하는 별도의 클래스가 필요하여 SESS HW State 클래스를 추가하였음을 기술하였다.

Fig. 2. Example of Highlighted Class Diagram with a Tactic

4.5 스텝 5. 정제된 아키텍처 평가

개요: 스텝 5에서는 택틱 적용으로 정제된 아키텍처가 스텝 1에서 고려된 모든 NFR을 잘 반영하였는지를 평가한다. 이 스텝에서의 평가 대상은 아키텍처 설계 시 고려된 NFR이 아키텍처에 모두 반영이 되었는지를 평가하는 것이므로, 각 산출물 간의 추적성 관계를 이용하여 아키텍처 설계에 누락된 NFR이 없는지를 확인한다. Fig. 3은 이전 스텝에서 나온 산출물 간의 추적성 관계를 보여준다.

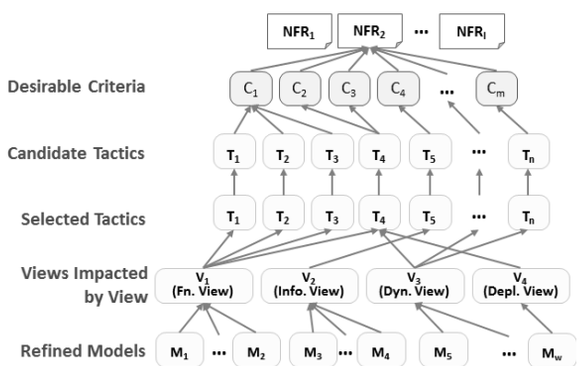

Fig. 3. Traceability Among Artifacts

스텝 1에서 요구사항 만족 기준들은 특정 NFR을 만족시킬 수 있도록 도출되었으므로, 다음과 같이 NFR_i과 C_j 간에는 *DeriveRel* 관계가 있다.

DeriveRel = {(NFR_i, C_j) | NFR_i로부터 C_j가 도출되었음.}

스텝 2에서 각 요구사항 만족 기준을 해결할 수 있도록 택틱들이 선정되므로, 다음과 같이 C_j와 T_k 간에는 *RealizedByRel* 관계가 있다.

RealizedByRel = {(C_j, T_k) | C_j는 T_k를 만족시키기 위해 선택됨}

스텝 3에서 택틱들은 특정 뷰에 적용되기 때문에, T_k와 V_x 간에는 *AppliedToRel* 관계가 성립되며, 스텝 4에서는 V_x에 속하는 특정 모델 M_z에 택틱 T_k를 적용하여 수정하므로, *RefinesRel* 관계가 성립된다.

AppliedToRel = {(T_k, V_x) | T_k는 V_x에 적용됨}

RefinesRel = {(T_k, V_x, M_z) | V_x에 속하는 M_z가 T_k 적용으로 정제됨}

이 추적성 관계를 기반으로 정제된 아키텍처 모델을 평가하면, 누락된 NFR이 없는지, 정확하게 NFR이 고려되었는지를 효과적으로 확인할 수 있다.

입/출력물: 이 스텝의 입력물은 스텝 1에서 스텝 4에서 작성한 모든 산출물이며, 산출물은 추적성 관계를 기반으로 아카텍처를 평가한 결과이다. 평가한 결과는 Fig. 3과 같이 그림으로 표현한다.

지침: 이 스텝은 이전 스텝에서 만든 산출물 간의 추적성 관계를 이용하여 주어진 NFR들이 모두 아키텍처 뷰 모델에 반영되었는지를 확인한다.

첫째, 각 뷰 (V_x)에 속한 모델 (M_z)에 적용된 택틱 (T_k)들을 먼저 확인한다. 즉, 각 뷰 모델 별로 NFR 적용으로 인해 변경된 부분을 파악하고, 이와 관련된 택틱들을 확인한다. 이 과정을 거쳐, 각 뷰에 택틱이 올바르게 적용되었는지도 평가할 수 있다.

둘째, 각 택틱이 만족시키는 요구사항 만족 기준 (C_j)를 확인한다. 이때 1) 모든 택틱이 만족 기준을 누락하지 않고 만족시키고 있는지를 확인하며, 2) 각 택틱이 만족 기준을 잘 만족시키고 있는지도 확인해야 한다.

마지막으로, 각 만족 기준이 설명하고 있는 비기능적 요구사항 (NFR_i)를 확인한다. 이 때도 마찬가지로, 1) 모든 만족 기준들이 요구사항 명세서에 정의된 모든 NFR을 완전하게 만족하고 있는지 확인하며, 2) 각 기준이 NFR을 잘 고려하여 도출되었는지도 확인해야 한다.

Fig. 3과 같은 그림을 이용하여, M_z, V_x, T_k, C_j, NFR_i 간에 유도 관계를 표현할 수 있다. 이런 그림을 이용하면, NFR을 설명하기 위한 만족 기준이 없는지, 각 만족 기준을 해결하기 위한 택틱이 누락되었는지 등을 쉽고 효과적으로 확인할 수 있다.

5. 사례 연구 - 소프트웨어 플랫폼 아키텍처 설계

본 장에서는 제안된 기법의 적용가능성 및 실용성을 평가하기 위해, 디지털 건강 평가 플랫폼인 RAINBOW의 아키텍처 설계에 적용한다. 소프트웨어 플랫폼은 단일 시스템보다 비기능적 요구사항이 복잡하기 때문에, 플랫폼 아키텍처 설계에 제안된 기법을 적용한다.

5.1 목표 시스템 개요

RAINBOW는 의료 컨텍스트를 이용한 여러 종류의 건강 평가 애플리케이션을 개발하고 운영하는데 필요한 공통 기능성을 제공하는 디지털 건강 분석 소프트웨어 플랫폼으로, 크게 다음과 같이 여섯 가지 종류의 기능을 제공한다.

- 사용자 프로파일 관리: 플랫폼과 상호작용하는 건강 분석 애플리케이션을 설치하여 사용하는 최종 사용자에 대한 프로파일을 등록, 수정, 검색, 삭제한다.
- 건강 분석 애플리케이션 메타 정보 관리: 건강 분석 애플리케이션 별 사용하는 건강 분석 컴포넌트를 추적하기 위해 애플리케이션 메타 정보를 등록, 검색, 수정, 삭제한다.
- 의료 데이터 관리: 애플리케이션으로 수집된 의료 데이터

를 저장하거나 요청에 맞게 검색한다.

- 건강 분석 컴포넌트 관리: 내장 (Built-in) 분석 컴포넌트와 플러그인 (Plug-in) 분석 컴포넌트를 관리하는 기능을 제공한다. 내장 분석 컴포넌트는 여러 애플리케이션에서 필요로 하는 공통적인 분석 기능을 플랫폼에서 자체적으로 제공하는 것이며, 플러그인 분석 컴포넌트는 플랫폼 내에서 제공되지 않고 애플리케이션 개발자가 필요에 따라 추가한 것이다. 분석 컴포넌트를 새롭게 런타임에 추가하여 실행시킬 수 있기 하기 위해, 분석 컴포넌트 등록, 검색, 수정, 삭제하고, 동적으로 추가된 분석 컴포넌트 로딩 (Loading) 및 언로딩 (Unloading)한다.
- 분석 컴포넌트 실행: 플랫폼 내의 내장 분석 컴포넌트 또는 플러그인 분석 컴포넌트를 실행하여 사용자의 건강 상태를 평가한다.
- 건강 분석 결과 관리: 분석 컴포넌트에 의해 예측 및 계산된 건강 분석 평가 결과를 저장, 검색, 삭제한다.

5.2 비기능적 요구사항

복수 개의 건강 평가 애플리케이션에게 고품질의 서비스를 제공하기 위해, RAINBOW 플랫폼은 크게 3가지의 비기능적 요구사항을 고려하여 설계된다.

- NFR #1. 추가적으로 확장될 IoT 건강 데이터 수용: 건강 평가 애플리케이션들로부터 수집되는 건강 데이터의 종류 및 형식이 다양하기 때문에, 이질적인 데이터를 효과적으로 관리해야 한다.
- NFR #2. 플러그인 컴포넌트와의 상호운영성 보장: 소프트웨어 플랫폼이 모든 종류의 건강 분석 기법을 제공하기 어렵기 때문에, 확장성 있게 다양한 건강 평가 컴포넌트를 수용하여 실행할 수 있도록 해야 한다.
- NFR #3. 건강 평가 결과의 정확성 보장: RAINBOW는 높은 정확성을 보장한 건강 평가 분석 결과를 산출하도록 설계되어야 한다.

5.3 제안된 기법을 적용한 결과

본 장에서는 제안된 프로세스가 실용적으로 정의되었는지를 확인하기 위해, NFR #2을 RAINBOW 플랫폼의 아키텍처 설계에 적용한 결과를 보여준다.

스텝 1 수행 결과: 스텝 1에서는 NFR #2를 구체적으로 이해하는데 도움이 되는 요구사항 만족 기준을 식별한다. NFR #2을 위해서, RAINBOW는 외부 건강 평가 애플리케이션 개발자에 의해 개발된 플러그인 건강 평가 컴포넌트를 실행시킬 수 있는 장치를 제공해야 한다. RAINBOW와 건강 평가 컴포넌트 간에는 서로 인터페이스를 통하여 상호작용을 하기 때문에, 인터페이스 호환성에 대한 잘 알려진 사실을 기반으로 요구사항 만족 기준을 식별할 수 있다. Fig. 4는 두 컴포넌트 또는 컴포넌트와 플랫폼 간 인터페이스 호환성을 판단할 때 고려되는 잘 알려진 Fact를 보여준다.

일반적으로 인터페이스는 오퍼레이션 이름, 입력 매개변수, 반환 타입을 기반으로 상호작용을 하므로, 인터페이스 명세가 서로 일치해야 한다. 그리고, 같은 언어 구현되거나 같은 환경에서 운영되는 컴포넌트만 호출된다.

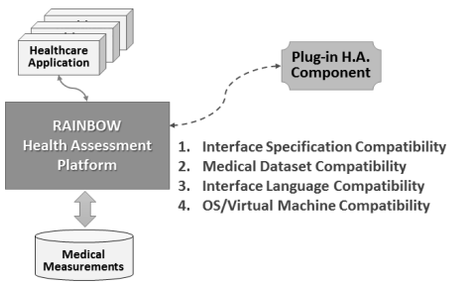

Fig. 4. Interoperability-related Facts for Plug-in Components

이 결과로, 컴포넌트 간 인터페이스를 통한 호환성에 관련된 잘 알려진 Fact를 기반으로 Table 4와 같이 NFR #2를 위한 4가지 요구사항 만족 기준을 도출한다.

Table 4. Four Desirable Criteria Derived from NFR #2

Criteria ID	Name	Description
C1	Maintaining Compatibility on Interface Specification	The interface of a plug-in HA component should be compatible with the required interfaces of RAINBOW platform, in terms of method signatures and value ranges of parameters.
C2	Maintaining Compatibility on Health Dataset	The medical dataset manipulated by a plug-in HA component should be compatible with the medical datatypes stored in RAINBOW platform.
C3	Maintaining Compatibility on Implementation Language	The implementation language for a plug-in component should be compatible with that of RAINBOW platform.
C4	Maintaining Compatibility on Operating Environment like OS and Virtual Machine	The operating system or the virtual machine needed to run HA components should be compatible with that of RAINBOW platform.

위의 4가지 요구사항 만족 기준은 Fig. 4의 컴포넌트 간 호환성에 대해 잘 알려진 Fact를 기반으로 도출된 것이다.

스텝 2 수행 결과: 스텝 2에서는 각 요구사항 만족 기준을 해결하는데 필요한 디자인 택틱을 선정한다. 이를 위해, 먼저 소프트웨어 설계 및 컴포넌트 설계에 관련된 문헌 등을 기반으로 각 요구사항 만족 기준을 위해 적용될 수 있는 후보 택틱들을 조사하였다. Table 5는 4가지 요구사항 만족 기준을 해결하기 위해 적용될 수 있는 후보 택틱 목록을 보여준다.

C1인 인터페이스 명세 관점에서 호환성을 유지하기 위해서 인터페이스 표준화와 관련된 택틱 (T1, T2)과 정의된 인터페이스 내에서 구현의 다양성을 지원하는 택틱 (T3, T4)들을 선정하였다. 그리고, 표준화된 인터페이스를 통해 플러그인 된 컴포넌트 실행을 지원하기 위한 택틱 (T5)를 선정하였다. C2인 건강 데이터 집합 관점에서 호환성을 유지하기 위해 변경 및 진화 가능성이 높은 건강 데이터와 건강 프로파일 데이터

Table 5. List of Candidate Tactics for RAINBOW

Criteria ID	Candidate Tactics
C1	T1. Define Required Interface Hierarchy
	T2. Apply Strategy Pattern
	T3. Apply Template Method Pattern
	T4. Apply the Concept of Dynamic Comp. Loading
C2	T5. Use Meta-level Data Types
C3, C4	T6. Support the Concept of Micro Service

종류에 대한 메타 레벨 엔티티를 추가하는 택틱 (T6)을 선정하였다. 마지막으로, 구현 언어와 운영 환경 관점의 호환성을 유지하기 위해 마이크로 서비스 개념을 적용하는 택틱 (T7)을 선정하였다. 마이크로 서비스는 RAIBNOW 외부에서 운영이 되고, 런타임에 RAINBOW와 상호작용을 통해 기능을 수행하기 때문에, RAINBOW와 구현 언어 및 운영 환경이 달라도 호출이 가능해진다.

그리고, 검색된 택틱들에 대하여 RAINBOW 플랫폼 아키텍처 설계에 적용할 경우에 장/단점, 충돌 여부 등을 분석하여 최종 택틱 목록을 선정하였다. Table 6은 7개의 후보 택틱의 적용 가능성을 분석한 결과 일부를 보여준다.

Table 6. Candidate Tactic Evaluation Results for RAINBOW

Tactic ID	Name	Decision (Y/N)	Criteria ID	Rationale
T1	Define Required Interface Hierarchy	Y	C1	**[In terms of Applicable Situation]** A standardized interface is known as one of the effective solutions for addressing an incompatibility issue. If required interfaces are defined as a form of hierarchy, developers can develop plug-in components by choosing the right interface in the hierarchy. **[In terms of Pros & Cons]** Developers may have development burdens since they have to follow the standard. However, this disadvantage overweighs its advantages.
T2	Apply Strategy Pattern	Y	C1	**[In terms of Applicable Situation]** Strategy pattern is well applied where there are variabilities on the method implementation. **[In terms of Pros & Cons]** With the strategy pattern, we can have diverse implementations for an operation, but also there is a performance penalty. By considering the RAINBOW execution environment, this problem may not be severe
…	…	…	…	…

Table 6의 분석 결과로 7개의 후보 택틱이 모두 최종 택틱으로 선정되었다. 나아가, 선정된 택틱들은 서로 상호 보완 관계에 있기 때문에 7개의 택틱을 모두 적용할 경우 발생할 수 있는 충돌 문제가 없어 택틱 적용으로 인한 장점이 극대화될 것으로 예상된다.

스텝 3 수행 결과: 스텝 3에서는 선정된 7개의 택틱이 이전에 작성한 아키텍처 뷰 모델에 어떻게 반영되는지를 분석한다. Table 7은 선정된 택틱을 적용할 경우의 뷰 변경 범위를 분석한 결과를 보여준다.

Table 7. Tactic Impact Analysis Result

View / Tactic ID	Functional View	Information View	Behavioral View	Deployment View
T1	Add required interface hierarchy for plug-in components to H.A. Comp. Executor.			
T2	Add strategy pattern-related classes to H.A. Comp. Executor.			
…	…	…	…	…

T1, T2, T3, T4는 주로 플러그인 건강 평가 컴포넌트의 실행에 필요한 설계와 관련되기 때문에, 기능 뷰의 건강 평가 컴포넌트의 클래스 구조, 건강 평가 컴포넌트 호출하는 오퍼레이션 등에 영향을 미친다.

스텝 4 수행 결과: 스텝 4에서는 스텝 3에서 분석한 내용을 이용하여, 각 아키텍처 뷰 모델을 변경한다. Fig. 5는 택틱 T1, T2, T3를 적용하여 설계한 H.A. Comp. Executor 컴포넌트의 내부 구조를 보여준다.

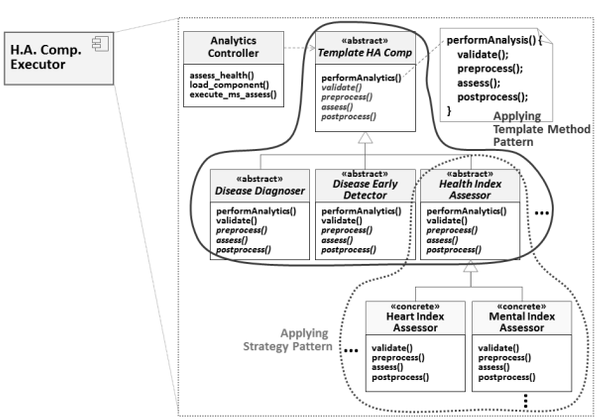

Fig. 5. Result of Applying T1, T2, and T3 - Internal Design of H.A. Comp. Executor

NFR을 적용하기 이전에는 H.A. Comp. Executor의 내부 구조에 대한 설계에 HA Comp 클래스만이 포함되어 있었

다. T1을 적용하여 HA Comp 의 계층 구조를 세부화하여 정의하였다. 이 계층 구조는 병원 의료진과 현재 가용한 건강 관련 애플리케이션에서 지원하는 건강 평가 방법에 분석 결과를 기반으로 작성하였다. T2, T3을 적용하여 각 건강 평가 방법 간의 공통성 및 가변성을 분석하여 인터페이스 계층 구조를 정제하였다. 최상위에 위치한 Abstract_HA_Comp의 하부에는 각각 다른 알고리즘으로 구현한 건강 평가 클래스들이 위치한다. IoT 디바이스를 이용한 건강 평가 방법들은 그 종류에 상관없이 입력 데이터 유효성 평가, 입력 데이터의 전처리, 분석, 분석 결과 후처리 단계로 이루어지지만, 각 단계를 처리하는 방법은 차이가 있다. 이를 위해, 템플릿 메소드 패턴을 이용하여 최상위 Template_HA_Comp 클래스를 정의하였고, performAnalytics() 오퍼레이션은 나머지 4개의 추상 메소드를 순차적으로 호출하도록 구현하였다. 그리고, 그 하위 클래스에도 구현 상의 차이가 존재할 수 있으므로, 전략 패턴을 적용하였다. 이렇게 함으로써, 플랫폼은 건강 분석 별로 최대한의 공통적인 알고리즘을 제공할 수 있고, 개발자는 필요로 하는 기능을 최대한 제공하는 클래스를 상속 받아 애플리케이션에 맞는 플러그인 건강 평가 기능을 구현할 수 있다. 그리고, 이들은 모두 Abstract_HA_Comp의 오퍼레이션을 제공하고 있으므로, Analytics Controller에 의해 호출이 가능하다.

이와 유사하게, 다른 택틱들을 적용하여 기능 뷰, 정보 뷰, 행위 뷰, 배치 뷰를 정제하였다. 이미 많은 설계 결정 사항은 스텝 3에서 이루어졌기 때문에, 이 결정사항을 설계 가이드라인으로 참고하여 보다 쉽게 아키텍처 뷰 모델들을 정제할 수 있었다.

스텝 5 수행 결과: 스텝 5에서는 각 스텝에서 나온 산출물 간의 추적성을 기반으로 모든 NFR들이 아키텍처 설계에 잘 반영이 되었는지를 확인한다. Fig. 6은 NFR #2인 플러그인 컴포넌트와의 상호운영성(Interoperability) 보장과 아키텍처 설계 모델 간의 추적성 관계를 보여준다.

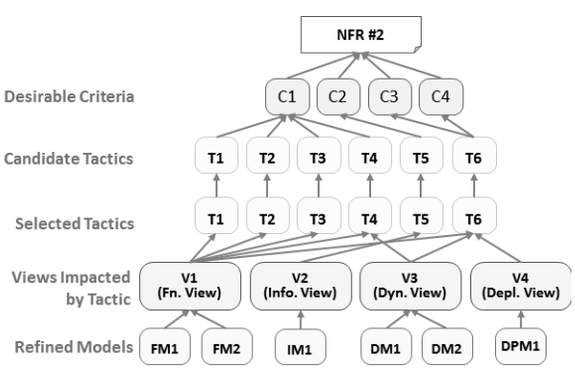

Fig. 6. Architecture Evaluation Result for NFR #2

NFR #2로부터 4개의 요구사항 만족 기준을 도출하였으며, 이 요구사항 만족 기준은 상호운영성에 대해 잘 알려진 사실로부터 유도되었다. 즉, 상호운영성을 만족시킬 수 있는 완전한 집합이 만족 기준으로 식별되었다. 4개의 요구사항 만족 기준을 해결하기 위해 총 6개의 택틱이 선정되었으며, 선정된 택틱은 4개의 모든 요구사항 만족 기준을 충족시킨다. 아키텍처 명세서에 포함된 4개의 뷰는 6개의 택틱이 잘 반영되었음을 확인할 수 있으며, 총 6개의 모델이 NFR #2를 만족시킬 수 있도록 정제하였다. 위의 절차를 통해 NFR #2를 잘 반영할 수 있도록 아키텍처가 정제되었음을 확인할 수 있다.

평가 및 결과: 설계 기법 및 설계 프로세스를 평가하는 것은 어려우므로, 본 논문에서는 사례 연구를 수행함으로써 제시된 프로세스의 적용성 및 실용성을 평가하였다. 사례 연구의 도메인은 건강 평가 소프트웨어 플랫폼으로 선정하였다. 선정 근거는 1) 단일 소프트웨어 보다 소프트웨어 플랫폼 아키텍처 설계 난이도가 매우 높으며, 2) 디지털 건강 평가 도메인은 IoT 기술 발전으로 새롭게 유망받고 있는 도메인이므로 기존 시스템과는 다른 요구사항을 가지기 때문이다.

이 시스템의 아키텍처 설계 결과 4가지 뷰를 표현하기 위하여 총 16개의 다이어그램과 이에 대한 설명이 포함된 70페이지 분량의 소프트웨어 아키텍처 명세서를 작성하였다. 제시된 프로세스를 따라 작성하여 보니, 복잡하고 난이도가 높은 NFR도 보다 체계적으로 아키텍처 설계에 반영할 수 있었다. 특히, 제시된 프로세스의 다음 특징이 소프트웨어 아키텍처 설계를 보다 효과적으로 수행할 수 있었다.

- 5개의 스텝으로만 구성된 프로세스: 제시된 프로세스는 많은 스텝을 포함하고 있지 않아, 적은 수의 스텝 수행으로 아키텍처를 설계할 수 있었다.
- 상세한 수행 지침 및 산출물 템플릿: 각 스텝에는 상세한 지침과 템플릿에 정의되어 있어, 각 스텝의 결과물을 만드는데 어려움을 줄일 수 있었다.
- 추적성 관계를 이용한 NFR 평가: 스텝 5에서 NFR이 아키텍처 설계에 모두 반영되었는지를 확인하기 위해 추적성 관계를 이용하였고, 이로 인해 누락된 NFR 없이 요구사항을 모두 충족하는 아키텍처를 설계할 수 있게 도움이 되었다.

6. 결 론

소프트웨어 아키텍처 설계에서 가장 중요한 점은 비기능적 요구사항(Non-Functional Requirement, NFR), 즉 품질 요구사항과 제약사항을 만족시키는 것이다. NFR을 고려한 아키텍처 설계 기법은 소프트웨어 아키텍처 설계 연구 분야에서 기술적 난이도가 높으며, 앞으로도 많은 연구가 필요한 주제 중 하나이다. 그러나, 목표 시스템에 특화된 비전형적인 NFR을 위한 설계 방법에 대한 연구는 많이 진행되고 있지 않고, 소프트웨어 아키텍트가 보유한 지식과 경험에 의해 비전형적인 NFR을 만족시킬 수 있는 효과적인 방법과 택틱을 유도하고 이를 기반으로 아키텍처를 설계한다. 그러므로, 비전형적인 NFR을 고려하여 아키텍처를 설계하는 효과적인 방법 및 택틱을 고안하는 것이 어렵다.

본 논문에서는 비전형적인 NFR을 만족시키는 소프트웨

어 아키텍처를 설계하는 효과적이며 체계적인 아키텍처 설계 방법론을 제안하였다. 이 방법론은 전형적인 NFR을 고려한 아키텍처 설계에도 적용될 수 있다. 제안된 방법론은 5개의 스텝으로 구성된 프로세스, 각 스텝에 대한 상세 활동 지침을 포함하였다. 그리고, 제안된 프로세스가 잘 설계되었음을 보이기 위해, 각 스텝의 산출물의 구성 요소를 나열하고, 산출물 간의 추적성(Traceability) 관계를 확인을 통한 검증 방법을 포함한다. 마지막으로, 제안된 방법론의 효율성과 실용성을 평가하기 위해 소프트웨어 플랫폼 아키텍처 설계 사례 연구를 수행한 결과를 제시하였다.

제안된 소프트웨어 아키텍처 설계 방법론의 특징은 다음과 같이 요약될 수 있다.

- 비전형적인 NFR을 위한 아키텍처 설계 프로세스를 제시함.
- 추적성을 고려한 체계적인 NFR 기반 아키텍처 평가 지침을 제시함.

제시된 방법론을 준수하여 아키텍처를 설계하면 비전형적인 NFR을 만족하는 고품질의 아키텍처를 보다 효과적으로 설계할 수 있을 것이라고 예상된다. 그리고, 추적성 관계를 활용하여 NFR을 충분히 반영하여 아키텍처를 설계하는데 도움이 될 것이라고 사료된다.

References

[1] M. Shaw and P. Clements, “The Golden Age of Software Architecture,” *IEEE Software*, Vol.23, No.2, pp.31-39, Mar.-Apr., 2006.

[2] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood, “Attribute-Driven Design (ADD), Version 2.0,” *Software Engineering Institute (SEI), TECHNICAL REPORT CMU/SEI-2006-TR-023, ESC-TR-2006-023,* Nov., 2006.

[3] L. Bass, P. Clements, and R. Kazman, “Software Architecture in Practice,” 3rd Ed., Addison-Wesley Professional, Oct., 2012.

[4] W. G. Wood, “A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0,” *Software Engineering Institute (SEI), TECHNICAL REPORT, CMU/SEI-2007-TR-005, ESC-TR-2007-005,* Feb., 2007.

[5] J. Scott and R. Kazman, “Realizing and Refining Architectural Tactics: Availability,” *Software Engineering Institute (SEI), TECHNICAL REPORT CMU/SEI-2009-TR-006, ESC-TR-2009-006*, Aug., 2009.

[6] N. Rozanski and E. Woods, “Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives,” 2nd Ed., Addison-Wesley Professional, Nov., 2011.

[7] S. T. Kim, D. K. Kim, L. Lu, and S. Y. Park, “Quality-driven Architecture Development using Architectural Tactics,” *The Journal of Systems and Software*, Vol.82, No.8, pp.1211-1231, Aug., 2009.

[8] A. Tsadimas, M. Nikolaidou, and D. Angonostopoulos, “Handling Non-functional Requirements in Information System Architecture Design,” In *Proceedings of 2009 Fourth International Conference on Software Engineering Advances (ICSEA 2009)*, pp.59-64, Sep., 2009.

[9] H. Reza and E. Grant, “Quality-Oriented Software Architecture,” *In Proceedings of International Conference on Information Technology: Coding and Computing (ITCC 2005)*, May, 2005.

[10] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “A Methodological Approach to Apply Security Tactics in Software Architecture Design,” In *Proceedings of 2014 IEEE Colombian Conference on Communications and Computing (COLCOM 2014)*, Jun., 2014.

[11] *Systems and Software Engineering –Architecture Description, ISO/IEC/IEEE 42010:2011*, Dec., 2011.

라 현 정

e-mail : hjla80@gmail.com

2003년 경희대학교 전자정보학부(학사)

2006년 숭실대학교 컴퓨터학과(공학석사)

2011년 숭실대학교 컴퓨터학과(공학박사)

2011년～2013년 숭실대학교 모바일 서비스 소프트웨어공학센터 연구교수

2013년～현 재 (주)스마티랩 연구소장

관심분야 : Software Architecture, Mobile Cloud Computing, Internet of Things Computing

김 수 동

e-mail : sdkim777@gmail.com

1984년 Northeast Missouri State University 전산학(학사)

1988년/1991년 The University of Iowa 전산학(석사/박사)

1991년～1993년 한국통신 연구개발단 선임연구원

1994년～1995년 현대전자 소프트웨어연구소 책임연구원

1995년～현 재 숭실대학교 소프트웨어학부 교수

관심분야 : Object-Oriented Modeling, Software Architecture, Internet of Things Computing, Machine Learning-based eHealthcare