KIPS Trans. Softw. and Data Eng.
Vol.12, No.7 pp.285~294
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2023.12.7.285

A Study on Efficient Application of Architectural Patterns by the Taxonomy of Software Requirements

Jong-Woo Choi† · Sang Yoon Min††

ABSTRACT

As software grows continuously in scale and complexity, the role of software architecture has become increasingly important across various industries. Although software architects often rely on their experience and intuition when designing such architecture, there is a variety of methodologies being researched for architecture design. However, these methodologies do not address the specific effects of applying multiple architectural patterns to a system or the sequence in which they should be applied. In this study, we explain the variation in architectural design results depending on the order in which the same set of architectural patterns is applied to a single system. Based on this phenomenon, we identify requirements for applying architectural patterns and propose a method of classifying the patterns to be applied. We also propose a prioritization process for requirements to efficiently apply the classified patterns in a specific order. Finally, we show a case study that prioritizing requirements based on architectural pattern types is beneficial for efficient software architecture design in terms of quality attributes.

Keywords : Requirements Taxonomy, Requirements Prioritization, Software Architectural Patterns, Software Architecture Design

소프트웨어 요구사항 분류체계를 이용한 효율적인 아키텍처 패턴 적용에 관한 연구

최 종 우† · 민 상 윤††

요 약

다양한 산업에 영향을 미치고 있는 소프트웨어의 규모와 복잡도가 높아지면서 소프트웨어 아키텍처의 역할이 매우 중요해지고 있다. 소프트웨어 아키텍트는 이러한 아키텍처를 설계할 때 종종 경험적 직관에 의존한다. 그럼에도 아키텍처 설계에 관한 방법론이 다양하게 연구되고 있으나, 시스템에 여러 아키텍처 패턴(architectural pattern)을 적용하는 구체적인 방법이나 적용 순서에 따른 효과에 관해서는 다루고 있지 않다. 본 연구에서는 먼저, 같은 아키텍처 패턴 세트(set)를 동일 시스템에 적용할 때, 적용 순서에 따른 아키텍처 설계 결과의 상이성에 관해 설명한다. 이러한 현상적 논리를 바탕으로 아키텍처 패턴 적용이 필요한 요구사항들을 사전에 식별하고, 적용할 패턴을 분류하는 방안을 제시한다. 또한 분류한 패턴들을 효율적으로 적용하기 위해 적용 순서를 정할 수 있도록 요구사항의 우선순위를 정하는 절차를 제안하며 사례연구를 통해 아키텍처 패턴 유형을 기반으로 요구사항을 우선순위화하는 것이 품질 속성 측면에서 효율적인 소프트웨어 아키텍처 설계에 도움이 됨을 보인다.

키워드 : 요구사항 분류체계, 요구사항 우선순위화, 소프트웨어 아키텍처 패턴, 소프트웨어 아키텍처 설계

1. 서 론

소프트웨어 아키텍처는 시스템을 추론하는 데 필요한 구조의 집합으로 비즈니스 목표와 최종 결과물인 시스템 사이에서 다리 역할을 한다[1]. 또한, 아키텍처는 추상적인 시스템을 구체적으로 표현하여 고객과 개발자를 포함한 이해관계자들 간에 의사소통을 더 명확하게 할 수 있도록 한다. 소프트웨어가 다양한 산업에 영향을 미치고, 소프트웨어의 규모와 복잡도가 증가하면서 소프트웨어 아키텍처의 역할이 매우 중요해지고 있다[1, 2].

소프트웨어 요구사항은 이해관계자들이 앞으로 만들어질 소프트웨어에 대해 기대하는 사항들로[3], 소프트웨어 아키텍처는 이러한 소프트웨어 요구사항을 기반으로 설계된다[1]. 소프트웨어 요구사항은 개발해야 하는 시스템의 특정 기준에 따라 우선순위가 정해질 수 있는데, 품질 속성 기반 아키텍처 설계 방법론(Attribute Driven Design)에서는 비즈니스 가치(business value, biz. value)와 아키텍처 영향도(architectural impact, arch. impact)에 따라 요구사항의 중요도를 설정하고 우선순위화 한다[1].

※ 이 논문은 2022년 소프트웨어공학연구회의 우수논문으로 "소프트웨어 요구사항 분류 모델의 성능 향상을 위한 불균형 데이터 처리에 관한 연구"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 한국과학기술원 소프트웨어대학원 석사
†† 비 회 원 : 한국과학기술원 전산학부 겸직교수
Manuscript Received : December 22, 2022
First Revision : April 10, 2023
Accepted : April 21, 2023
* Corresponding Author : Sang Yoon Min(sang@sol-link.com)

아키텍처 패턴은 경험이 많은 아키텍트들의 지식을 집약하고 정형화한 솔루션으로 아키텍처 설계 시에 도움을 줄 수 있다[4]. 여러 종류의 아키텍처 패턴이 각 요구사항별로 관련될 수 있는데, 어떤 요구사항을 먼저 반영하는지에 따라 소프트웨어 아키텍처가 달라지며 이는 소프트웨어 품질과 연결된다. 소프트웨어 아키텍트는 소프트웨어 요구사항을 기반으로 아키텍처를 설계하는 데 종종 아키텍트의 경험적 직관에 의존한다[5]. 그럼에도 소프트웨어 아키텍처 설계 방법론이 많이 연구되고 있으며, 특히 패턴 기반 설계를 위해 아키텍처 패턴을 소개하는 서적들과 연구들이 다수 존재한다[1, 6, 7]. 또한 시스템에 적합한 아키텍처 패턴을 선택하기 위한 연구들이 있으며[8-10], 최근에는 기계학습(Machine Learning)을 이용해 아키텍처 패턴을 선택하는 연구가 진행되고 있다[4].

실무에서는 시스템을 설계할 때 요구사항 수가 매우 많아서 하나의 아키텍처 패턴으로 시스템을 구성하지 않고, 다양한 종류의 패턴을 적용하여 시스템을 구성한다[11]. 패턴들이 비효율적으로 적용되어 만들어진 아키텍처는 이후 개발 과정에서 다른 형태로 추가적인 노력이 필요해 품질 비용이 올라가고, 간혹 임시방편의 코드 수정으로 인해 개발 프로세스 후반부에 아키텍처가 변경되어야 하는 문제가 생기기도 한다. 이는 품질 비용을 높이게 되기 때문에 실무에서는 효율적인 아키텍처 패턴 적용을 매우 중요하게 여기고 있다. 그러나 기존 패턴 연구나 관련 서적에서는 각각의 패턴에 관해 설명하거나 적합한 패턴을 선택하는 과정을 설명하지만, 여러 개의 패턴을 적용하는 구체적인 방법이나 적용 순서에 따른 효과에 대해서는 다루고 있지 않다.

본 연구에서는 먼저 같은 아키텍처 패턴 세트(set)를 동일 시스템에 적용할 때 적용 순서에 따라 아키텍처 설계 결과가 달라짐을 보인다. 이러한 현상적 논리를 바탕으로 패턴 적용이 필요한 요구사항들을 사전에 식별하고, 적용할 패턴을 분류하는 방안을 제시한다. 그리고 분류한 패턴들을 적용하는 순서를 정하기 위해 체계적으로 요구사항을 우선순위화하는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서 아키텍처 패턴에 관한 연구와 서적, 패턴 적용의 실제, 요구공학에서의 요구사항 분류체계 연구 등 관련 연구에 관해 기술한다. 3장에서는 패턴 적용 순서의 차이를 보이고, 패턴 기반 요구사항 분류체계에 관해 설명한다. 4장에서는 사례연구를 통해 3장에서 제안한 방법론의 실제 효과에 관해 설명하고, 5장에서 결론과 후속 연구 방향을 제시한다. 이를 통해 아키텍트가 가장 효율적인 방법으로 요구사항의 반영 순서를 체계적으로 결정하여 패턴 적용 시 아키텍처를 일관성 있게 잘 관리하는 데 도움을 주고자 한다.

2. 관련 연구

2.1 아키텍처 패턴과 적용에 관한 연구

아키텍처 패턴을 소개한 서적들과 이를 적용할 때 생기는 문제에 관한 연구들이 있다. Bass et al.[1]은 일반적인 컨텍스트(context)에서의 문제를 해결하는 방안을 서술하는데 문서화된 패턴만으로는 특정한 상황에 적용하기가 부족하다고 보고 있다. 이를 위해 품질 속성 설계 전술(tactic)을 이용해 확장하는 것을 제시하고는 있으나 여러 품질 속성 설계 전술이나 여러 아키텍처 패턴을 같이 적용하는 방법은 기술하지 않고 있다.

Richards et al.[7]은 프로그램의 특성과 동작을 정의할 때 아키텍처 패턴을 이용하는데, 비즈니스 요구사항과 목표에 맞는 패턴을 선택하려면 각 아키텍처 패턴의 특성과 강점 및 약점을 알아야 한다고 설명하고 있다. 그러나 여러 아키텍처 패턴을 기술하고 주의할 점을 설명할 뿐 여러 패턴을 적용하는 방법은 보이지 않고 있다.

Kim et al.[12]은 일반적으로 특정 문제에 대한 패턴의 적용 가능성을 결정하는 것은 개발자가 패턴에 대해 가지고 있는 지식과 경험에 크게 의존하게 되어 패턴의 사용이 제한되므로 이 문제를 해결하기 위해 패턴의 문제 영역을 엄격하게 지정하는 방법을 제안하고 있다.

Kim et al.[13]은 요구사항의 문제를 분석하는 문제틀 방법과 해법을 제시하는 아키텍처 패턴을 매칭(matching)하는 방법을 제안하고 있으나 구체적으로 적용할 패턴을 도출하는 방법은 제시하지 않고 있다.

2.2 요구사항 분류에 관한 연구

Pandey et al.[14]에 따르면 요구공학에서 일반적으로 요구사항은 기능 요구사항과 비기능 요구사항으로 분류되고 비기능 요구사항은 다시 제약사항과 성능, 신뢰성 등의 품질 속성으로 분류된다. 이와 유사하게 공공 소프트웨어 사업 요구사항 작성 가이드라인에서는 요구사항을 크게 기능 요구사항, 비기능 요구사항, 기타 항목으로 분류한다. 그리고 비기능 요구사항을 성능 요구사항, 인터페이스 요구사항, 품질 요구사항, 제약사항 등으로 분류하고, 기타 항목을 유지관리 수행, 유지관리 인력 등으로 분류하고 있으나 아키텍처 설계를 고려하거나 적용을 고려한 분류체계는 제시하고 있지 않다[15].

3. 접근 방법

3.1 아키텍처 패턴 적용 순서에 따른 아키텍처

본 논문에서는 먼저, 관련 연구들에서 보이지 않은 아키텍처 패턴의 적용 순서에 따른 영향에 관해 확인한다. 동일한 패턴의 세트를 다른 순서로 적용하는 방법을 이용해 최종 아키텍처가 달라질 수 있음을 보인다. 실무에서 자주 사용되는 패턴들 중 3가지를 다른 순서로 적용하면서 클래스(class)의 위치가 어떻게 변화하는지 확인한다. 패턴은 Pipe-and-Filter 패턴, MVC(Model-View-Controller) 패턴, Layered 패턴 3가지를 이용하고 임의의 A, B, C 클래스를 활용한다. 각각의 클래스는 가공 전 데이터, 가공 중간 데이터, 가공 완료된 데이터에 관련된 클래스라고 가정한다. Layered 패턴과 MVC 패턴은 모듈 뷰(module view)와 관련되고, Pipe-and-Filter

패턴은 C&C 뷰(Component and Connector view)와 관련 있지만 패턴 적용 순서에 의해 변화되는 클래스 간 관계를 살펴보기 위해 혼용된 뷰를 이용하기로 한다.

먼저, Pipe-and-Filter 패턴, Layered 패턴, MVC 패턴의 순서로 적용해 아키텍처를 설계해본다. 데이터 흐름을 위한 Pipe-and-Filter 패턴을 먼저 적용하고 데이터 가공 위치에 따라 관련된 A, B, C 클래스를 연결한다. 그 후에 Layered 패턴과 MVC 패턴을 같이 적용하면 Fig. 1과 같이 설계된다. 두 번째로 Layered 패턴, Pipe-and-Filter 패턴, MVC 패턴의 순서로 적용하여 설계하면 Fig. 2와 같이 설계된다. 먼저 A, B, C 클래스를 계층화한 후에 데이터 흐름을 위한 Pipe-and-Filter 패턴을 적용한다. A, B, C 클래스가 파이프라인(pipeline)으로부터 각각 관련 있는 데이터에 대한 정보를 수신하기 위해 프록시(proxy) 형태의 중간 클래스가 필요하게 된다. 또한 MVC 패턴을 적용하면서 A, B, C 클래스의 데이터를 제어하거나 뷰에 보여줘야 할 때도 프록시 형태의 중간 클래스가 필요하게 된다. 세 번째로 MVC 패턴, Pipe-and-Filter 패턴, Layered 패턴의 순서로 적용하면 Fig. 3과 같다. Fig. 1과 동일한 결과가 설계되었는데 패턴의 적용 순서가 바뀔 때 같은 결과물도 나올 수 있음을 알 수 있다.

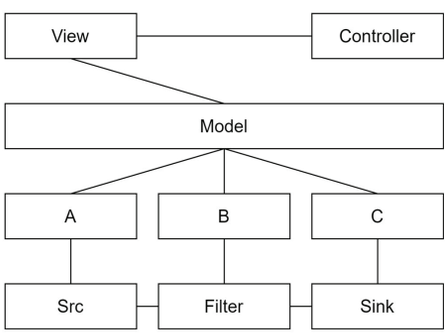

Fig. 1. Architecture Designed by Applying Pipe-and-Filter, Layered, and MVC Patterns in Order

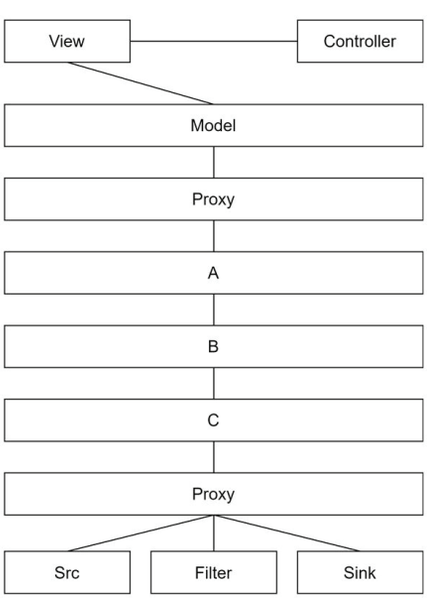

Fig. 2. Architecture Designed by Applying Layered, Pipe-and-Filter, and MVC Patterns in Order

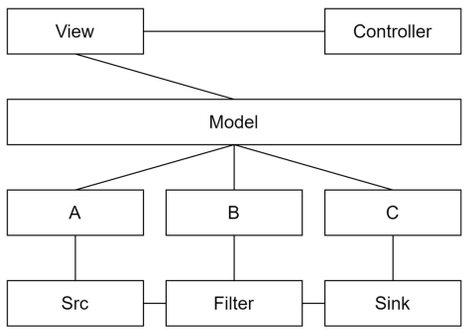

Fig. 3. Architecture Designed by Applying MVC, Pipe-and-Filter, and Layered Patterns in Order

세 가지 패턴의 순서를 변화시키면서 적용해 본 결과 최종 아키텍처가 다르게 나올 수 있음을 확인할 수 있다. 특히 Fig. 2의 최종 아키텍처가 부적절해 보이지만 실무에서 프레임워크 설계 초기에 데이터 마샬링(marshalling)과 보호를 위한 중간 추상화 계층을 생성하면서 나타난 실제 사례이다. 아키텍트가 아키텍처 패턴을 적용하면서 경험적 직관으로 적용 순서를 바꿀 수 있으나, 체계적인 접근 관점에서 보면 패턴 적용 절차가 효율적인 아키텍처 설계에 도움이 될 수 있음을 알 수 있다.

3.2 패턴 기반 요구사항 분류체계

패턴 적용이 필요한 요구사항을 식별하고 관련 패턴별로 분류하는 방안을 제시한다. 그리고 분류한 패턴을 기반으로 요구사항을 체계적으로 우선순위화하고, 패턴을 순서대로 적용하는 절차를 제안한다. 본 논문에서는 이 절차를 패턴 기반 요구사항 분류체계(Pattern-Based Requirements Taxonomy, PRT)라고 정의한다. PRT 방법론을 설명하기에 앞서 패턴 분류에 활용할 수 있는 소프트웨어 아키텍처 패턴 풀(pool)을 새롭게 정리한다. Bass et al.[1]가 기술한 패턴 카탈로그(catalog)를 참고하여 실무에서 주로 사용하는 패턴별로 적용 속성 키워드(keyword)를 추출한다. 키워드는 TextFixer[1]의 Word Counter를 통해 패턴 카탈로그에 등장하는 단어의 빈도수를 이용해 추출한다. 이때 불용어(stopwords)는 제외하고, ‘software’와 같이 문서에서 특정 패턴만을 위한 설명이 아닌 단어와 해당 패턴의 단점이나 약점을 설명하는 부분의 단어는 제외한다. 추출한 키워드의 빈도수 분석 예시는 Table 1과 같다. Table 1은 layered pattern 설명에 대한 단어별 빈도수의 예시로 단어 리스트의 일부분을 보여준다. 아키텍처 패턴 별 키워드를 추출하여 정리한 소프트웨어 아키텍처 패턴 풀은 Table 2와 같다.

1) https://www.textfixer.com.

Table 1. Keyword Frequency Analysis for the Layered Pattern

Primary Keywords	Frequency	Common Words	Frequency
layer	11	the	25
layers	7	a	19
modules	6	of	17
system	6	to	13
allowed	4	in	11
software	4	and	10
layering	3	this	7
partition	2	is	7
separation	2	that	6
independently	2	be	5

Table 2. Software architecture pattern pool

Pattern Name	Keywords
Layered	layer, modules, bridging, nonadjacent, relation, strict, partition, called, modifiability, separation, independently, portability, public, abstractions, complexity, uses, interface, interact, relationship, cohesive, grouping, segmented, undirectional and etc.
Broker	broker, service, server, client, provider, failure, distributed, complex, component, location, dynamically, proxy, communication, messaging, marshaling, compatible, requests, return, availability, interoperate, exchange, deliver, bindings, intermediary, unmarshaling, protocol, indirection and etc.
Client-Server	server, client, service, request, reply, web, protocol, data, resources, ports, access, interactions, calls, browser, scalability, factoring, connection, distributed, availability, multiple, invoke, change, callback, notification, waiting, send, receive, request, host and etc.
Publish-Subscribe	event, consumer, producer, data, subscribe, bus, connector, interact, messages, publisher and etc.
Peer-to-Peer	peer, service, data, distributed, availability, interaction, request, response, interact, decentralized, delivery, protocol, security, consistency, backup, routing, p2p, scalability, indexing, routing, capability, flexible, seprated, discovery and etc.
Model-View-Controller (MVC)	user, data, interface, view, model, controller, separate, interaction, manages, modifications, notification, changes, widgets, event, callback and etc.
Pipe-and-Filter	data, filter, pipe, input, output, processing, independent, ports, interaction, streams, transformation, downstream, sequence, concurrently, coupled, graph, passing, protocol, upstream, buffer, parallelization and etc.
Service-oriented architecture (SOA)	service, soa, consumer, provider, distributed, performance, registry, modifiability, security, interoperability, shared, request, transaction, participants and etc.
Shared-data	data, shared, store, accessor, consumer, interaction, transaction, database, and etc.
Map-reduce	map, data, sorting, key, partition, instances, parallel, repository, nodes and etc.
Multi-tier	tier, multi, allocation, adjacent, grouping, eventbased, sharing, notification and etc.

Fig. 4에서처럼 PRT 방법은 체계적으로 패턴을 분류하고 적용하는 절차를 보인다. Fig. 4에서 타원은 PRT의 시작과 끝을 나타내고 둥근 사각형과 마름모는 행위와 결정조건을 나타내고, 평행사변형은 입력물 또는 산출물을 나타낸다. Fig. 4를 기반으로 PRT를 단계별로 상세히 설명한다.

1) 단계 1: 요구사항 식별 및 패턴 분류

전체 요구사항 리스트(Fig. 4의 All requirements)에서 패턴 적용이 필요한 요구사항을 식별하는 절차(Fig. 4의 Identify requirements)이다. 요구사항 문장에서 Table 2에 기술되어 있는 패턴별 적용 속성 키워드와 동일하거나 유사한 단어가 있는지 분석(Fig. 4의 Classify patterns with pattern keywords)하고, 해당 단어가 존재하면 패턴명을 후보자 패턴(candidate pattern)으로 작성해둔다. 모든 요구사항에 대해 이와 같은 작업을 반복하고 해당하는 단어가 없으면 적용할 패턴이 없는 것으로 간주한다.

2) 단계 2: 패턴의 우선순위 설정

분류한 패턴을 적용하기 전에 우선순위를 정하는 절차(prioritize)이다. 분류한 패턴들을 아래 Table 3의 기준(criteria)으로 나열한다. Table 3에서 외부 관계(external relationships)는 시스템 사용자나 외부 시스템, 또는 외부 모듈과 상호작용(interaction)하는 관계를 의미한다. 내부 관계(internal relationships)는 모듈 내부에서 모듈 내부 구성 요소 간의 관계를 의미한다. Bass et al.[1]에 보면 여러 아키텍처 뷰를 통해서 구성 요소의 구조와 구성 요소의 관계를 명확히 하고, Gamma et al.[16]에 기술된 설계 원칙을 보면 외부 구성 요소 간 관계를 먼저 설계하면서 결합도(coupling)를 낮추고 응집도(cohesion)를 높이고 있어 우선순위 기준으로 적합하다.

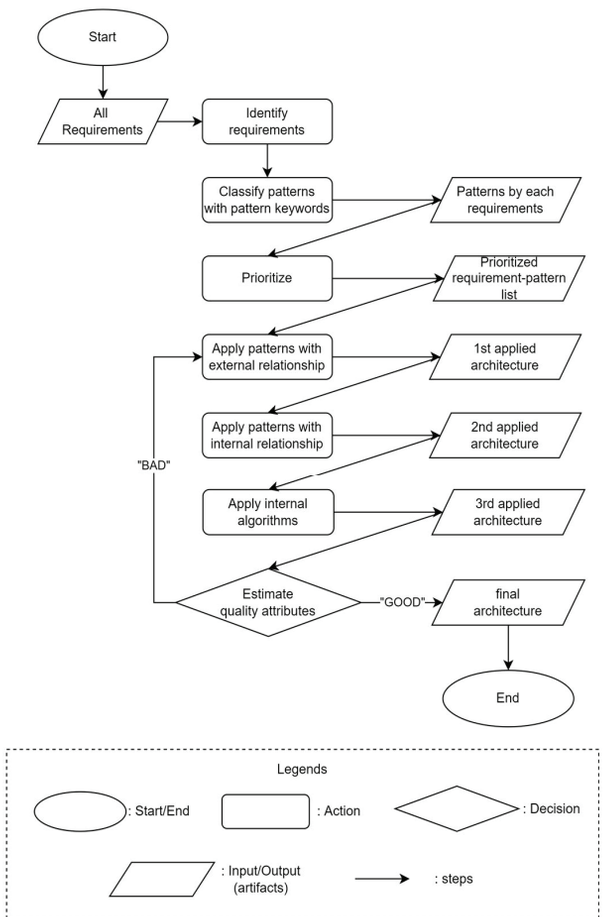

Fig. 4. Pattern-Based Requirements Taxonomy Flow

Table 3. Example of Prioritization of Candidate Pattern

Priority	Criteria	Pattern
Priority 1	Patterns with external relationship	Client-Server(4), Broker(3)
Priority 2	Patterns with internal relationship	Layered(2)
Priority 3	Internal algorithm that is difficult to pattern	None(1)

외부 관계와 연관된 패턴을 Priority 1로 하고 내부 관계와 패턴을 Priority 2로 설정 후에 패턴 적용이 쉽지 않은 구조 추가가 필요한 요구사항은 Priority 3으로 설정한다. 이때 Priority 1, Priority 2에서 분류된 패턴이 반복해서 등장하면 발생 빈도수를 패턴과 같이 작성하고, 이를 기반으로 요구사항을 우선순위화한다. Table 3의 Priority 1에서 Client-Server (4)는 패턴을 분류하는 동안 Client-Server 패턴이 4번 발생했다는 의미이다.

본 연구에서는 같은 우선순위 내에서는 패턴의 발생 빈도수를 기반으로 추가적인 우선순위를 결정하지만, 필요에 따라 다른 속성을 조건으로 사용할 수도 있다. 우선순위화된 자세한 요구사항 예시는 사례연구를 참고한다.

3) 단계 3: 외부 관계와 연관된 패턴 적용 (Priority 1)

외부 관계와 연관된 패턴은 물리적 단위와 관련이 있어 상대적으로 규모가 큰 패턴으로 판단된다. 분류된 패턴 중 등장 빈도수가 가장 높은 패턴을 적용하고, 해당 패턴과 같이 후보자로 등록된 패턴들은 바로 적용하지 않는다(Fig. 4의 Apply patterns with external relationship). 다만 후보자 중 요구사항에서 단독으로 도출된 패턴이 있으면 빈도수가 적어도 적용을 고려한다.

4) 단계 4: 내부 관계와 연관된 패턴 적용 (Priority 2)

외부와의 관계를 통해 물리적으로 분리된 이후에 개별 노드별 내부 관계에 따라 적용해야 하는 패턴들이다. 등장 빈도수가 가장 높은 패턴부터 적용하고 해당 패턴과 같이 후보자로 등록된 패턴들은 바로 적용하지 않는다(Fig. 4의 Apply patterns with internal relationship). 다만 후보자 중 요구사항에서 단독으로 도출된 패턴이 있으면 빈도수가 적어도 적용을 고려한다.

5) 단계 5: 내부 알고리즘 구조 적용 (Priority 3)

알려진 패턴으로 적용이 어려운 요구사항들을 적용하는 순서이다(Fig. 4의 Apply internal algorithms). 내부 알고리즘을 구현하기 위한 구조를 추가로 설계한다.

6) 단계 6: 아키텍처 평가

패턴들과 내부 구조가 반영되어 설계된 아키텍처를 각 시스템에서 요구하는 주요 품질 속성에 대해 평가한다(Fig. 4의 Estimate quality attributes). 평가는 각 품질속성 별 평가지표(metrics)를 이용해 측정한 후 평가 결과가 좋으면 최종 아키텍처로 사용하고, 평가 결과가 좋지 않으면 단계 3에서 적용되지 않은 후보 패턴을 선택하면서 단계 3부터 다시 진행한다.

4. 사례 연구

사례연구를 통해 PRT 방법론의 효용성을 확인하기 위해 Github에 공개된 open webOS의 uMediaServer 모듈을 실험 대상 소프트웨어로 선정한다. open webOS는 리눅스 커널에서 구동되는 모바일 운영체제로 현재 스마트TV와 같은 스마트 장비에서 웹 중심의 플랫폼으로 이용된다. 선정한 uMediaServer는 open webOS에서 미디어 프레임워크의 중심을 담당하는 모듈이다[17]. 주요 기능은 미디어 재생과 관련된 인터페이스를 제공하고, 미디어를 재생할 수 있는 파이프라인들의 우선순위를 스케줄링하는 역할을 가지고 있다.

4.1 실험 대상 소프트웨어 요구사항

요구사항 패턴 분류를 위해 반영할 요구사항을 정리한다. 실험 대상 소프트웨어의 주요 요구사항은 Table 4와 같다. 각 요구사항을 비즈니스 가치와 아키텍처 영향도에 따라 중요도를 설정한다. H는 High, M은 Middle, L은 Low를 나타낸다. 일반적으로 중요도를 바탕으로 요구사항을 우선순위화한다.

Table 4. Software Requirements of Sample Software

No.	Requirements	(Biz. value, Arch. impact)
1	uMediaServer should be able to deliver media commands (e.g. play, pause and etc.) to the video playback pipeline designated by the media application.	(H, M)
2	uMediaServer should be able to deliver commands (e.g. startRecord, stopRecord and etc.) to the camera pipeline specified by the camera application.	(H, M)
3	Pipelines are separate and shouldn't be called directly from the application.	(M, M)
4	uMediaServer must deliver notification events and errors related to playback.	(H, M)
5	Multiple applications should be able to send play commands simultaneously.	(L, M)
6	uMediaServer should prioritize applications that can be played.	(M, H)
7	The communication system should be abstracted.	(L, M)
8	uMediaServer should always be running.	(M, M)
9	uMediaServer should provide applications with a separate communication library in the form of a proxy.	(L, H)
10	All command requests should be returned within 10msecs.	(M, H)

uMediaServer는 미디어 애플리케이션(application)에게 webOS의 미디어 파이프라인을 이용할 수 있도록 인터페이스를 제공하고, 현재 재생을 시작하는 컨텐츠의 정보를 애플리케이션에게 전달하는 기능을 담당한다. 또한, 사용자가 여러 애플리케이션을 바꿔가면서 다양한 미디어 컨텐츠를 재생하는 데 한정된 하드웨어 자원을 할당받아서 재생을 시작할 파이프라인을 스케줄링하는 기능이 중요하다.

애플리케이션과 파이프라인을 매핑(mapping)하고 명령어와 이벤트를 전달하기 때문에 명령 전달이 실패하면 문제가 생길 수 있어 가용성(availability)이 가장 중요하다. 또한, 인터페이스 변경 시 애플리케이션과 파이프라인에 영향을 줄 수 있어 변경 용이성(modifiability)이 중요하다. 다음으로 미디어 재생에 지연(delay)이 있으면 안되므로 명령 전달 성능과 스케줄링 성능이 그 다음으로 중요할 것으로 판단된다. 모든 품질속성을 만족하는 아키텍처 설계는 쉽지 않고, 각 품질속성은 서로 트레이드오프(trade-off) 관계가 있기 때문에 [1] uMediaserver의 주요 품질속성은 가용성과 변경 용이성, 성능으로 설정한다.

4.2 PRT 미적용 아키텍처 및 평가

1) 실험 대상 소프트웨어 아키텍처

PRT 적용 아키텍처와 비교하기 위해 Github에 공개된 소스코드를 역공학(reverse engineering)으로 다이어그램을 도출하면 Fig. 5와 같다. 역공학 도구 없이 직접 분석한 소스 코드 기반으로 작성한 다이어그램이다. 공개된 링크에 별도 설계 문서가 없어 다이어그램 기준으로 보았을 때 Broker 패턴이나 Client-Server 패턴을 사용한 것으로 판단된다. 애플리케이션에 이벤트 전달을 위해 Publish-Subscribe 패턴이 적용될 수 있으나 uMediaserver에는 콜백함수(callback function) 형태로 구현되어 있어 별도로 다이어그램에 나타나지는 않는다.

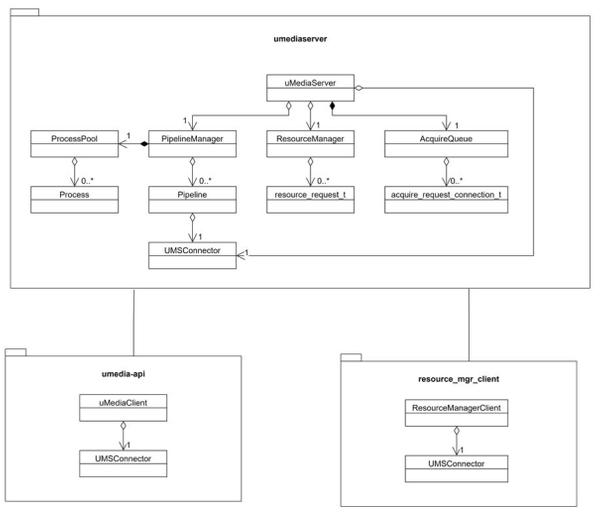

Fig. 5. The Architecture of Sample Software

2) 실험대상 소프트웨어 평가

uMediaserver의 품질 속성 중 가용성의 평가지표는 Equation (1)과 같이 평균 실패 시간(mean time between failure, MTBF)과 평균 보수 시간(mean time to repair, MTTR)을 이용하는 데[1], 이는 실제 소프트웨어를 실행시킨 후에 운영을 하면서 측정할 수 있는 런타임(runtime) 평가지표라 본 논문에서는 가용성 평가는 제외하고, 변경 용이성과 성능에 대해서만 평가한다. 참고로 Equation (1)은 0부터 1 사이의 값을 가지며 1에 가까울수록 가용성이 좋은 것으로 판단한다.

$$Availability = \frac{MTBF}{(MTBF+MTTR)} \quad (1)$$

변경 용이성 평가는 Equation (2)와 같이 특정 상황에서 변경이 필요할 때 수정되는 클래스 수를 평가지표로 사용한다. 0부터 1사이의 값을 가지며 1에 가까울수록 변경용이성이 좋은 것으로 판단한다. n은 수정하려는 항목 수이고, #은 개수를 의미한다.

$$Modifiability = \frac{1}{n}\sum_{1}^{n}(1 - \frac{\#\ of\ modified\ classes}{\#\ of\ whole\ classes}) \quad (2)$$

성능 평가는 Equation (3)과 같이 하나의 명령 전달 시 실행되는 클래스 객체의 수와 버퍼링(buffering) 로직을 가진 프로세스 수를 이용해 평가한다. 스마트TV는 메모리가 작은 임베디드 환경으로, 실무에서 모듈의 메모리 사용량과 버퍼링으로 인한 메모리 과다 사용으로 성능 저하가 발생한 실제 사례들이 있다. 클래스 객체의 수가 증가하면 메모리 사용량이 증가하고 이는 성능에 좋지 않은 영향을 줄 수 있다. 또한

Table 5. The Measurement Result of Original Umediaserver

Module	Modifiability	Performance
uMediaserver (original)	0.35 # = {3, 4}	0.34 # = {14, 1}

보통 버퍼링의 크기에 따라 성능이 달라질 수 있지만 버퍼는 메모리를 증가시키는 원인 중 하나로 성능에 영향을 주는 요소이다. 이로 인해 클래스 객체의 수와 버퍼링 로직을 가진 프로세스 수를 이용해서 성능을 평가한다. 0부터 1사이의 값을 가지며 1에 가까울수록 성능이 좋은 것으로 판단한다. n은 실행하려는 항목 수이고, #은 개수를 의미한다.

$$Performance = \frac{1}{2}\{\frac{1}{n}\Sigma_1^n\left(1-\frac{\#\ of\ touched\ classes}{\#\ of\ whole\ classes}\right)+\left(1-\frac{\#\ of\ processes\ having\ buffer}{\#\ of\ whole\ processes}\right)\} \quad (3)$$

Fig. 5에서 클라이언트(umedia-api)와 서버(umediaserver)가 분리되어 있어 개발자 간 업무 분리가 용이하고, 계층이 분리되어 통신체계(UMSConnector) 변경 시 수정이 용이한 면이 있다. 다만 브로커(broker) 형태로 uMediaServer가 단독 프로세스로 모든 umedia-api와 resource_mgr_client의 상태를 관리해 복잡도가 높고, 하나의 resource_mgr_client 관리를 위해 최대 4개의 Controller (ProcessPool, PipelineManager, ResourceManager, AcquireQueue)에 등록되어 메모리 낭비가 있다. 미디어 인터페이스를 추가하거나 수정할 때 uMediaClient, uMediaServer, Pipeline Manager, Pipeline 4개의 클래스 변경이 일괄적으로 필요하고, 파이프라인 우선순위 정책 변경 시에는 Resource Manager, resource_request_t, AcquireQueue 3개의 클래스 수정이 필요해 이러한 부분은 코드 변경이 용이하지 않을 것으로 분석된다. 추가적으로 성능 측면에서 보면 미디어 명령 전달 요청 시에 항상 uMediaServer를 통해서 전달되므로 성능에 좋지 않은 영향을 줄 수 있고, AcquireQueue 내부 버퍼로 인해 블록킹(blocking)되는 명령이 있을 경우 이후에 전달되는 이벤트 처리는 지연되게 된다. 평가 지표를 이용해 측정한 결과는 Table 5와 같다. 변경 용이성(Modifiability)은 미디어 API 수정과 스케줄링 정책 수정 건으로 측정을 해 0.35로 평가되고, 성능(Performance)은 미디어 API 실행과 버퍼링 프로세스 수에 대한 건으로 측정해 0.34로 평가된다.

4.3 PRT를 적용한 아키텍처 설계 및 평가

1) 단계 1: 요구사항 식별 및 패턴 분류

실험 대상 소프트웨어 요구사항 리스트인 Table 4를 바탕으로 요구사항별로 패턴 풀의 적용 속성 키워드에서 유사 단어를 찾아 구문에 표시하고 Table 6과 같이 후보자 패턴(candidate pattern)에 해당 패턴명을 추가한다. 대표적으로 요구사항 1번에서 “deliver”라는 단어가 패턴 풀에서 Broker 패턴, Peer-to-Peer 패턴의 키워드 리스트에 존재하고 유사한 단어로 명령을 전달한다는 의미에서 “send”도 유사하게 판단되어 Client-Server 패턴도 추가한다. 이렇게 키워드를 기반으로 모든 요구사항에 대해 연관 패턴을 찾아서 분류한다.

Table 6. Candidate pattern by sample software requirements

No.	Requirements	Candidate pattern
1	uMediaServer should be able to *deliver* media commands (e.g. play, pause and etc.) to the video playback pipeline designated by the media application.	Client-Server Broker Peer-to-Peer
2	uMediaServer should be able to *deliver* commands (e.g. startRecord, stopRecord and etc.) to the camera pipeline specified by the camera application.	Client-Server Broker Peer-to-Peer
3	Pipelines are *separate* and *shouldn't be called directly* from the application.	Layered MVC
4	uMediaServer must *deliver events* and errors related to playback.	Client-Server Broker Peer-to-Peer Publish-Subscribe MVC
5	uMultiple applications should be able to *send* play commands *simultaneously.*	Client-Server Peer-to-Peer
6	uMediaServer should prioritize applications that can be played.	None
7	The communication system should be *abstracted.*	Layered
8	uMediaServer should always be running.	None
9	uMediaServer should provide applications with a *separate communication* library in the form of *proxy.*	Broker Layered Peer-to-Peer MVC
10	All command *requests* should be *returned* within 10msecs.	Client-Server Broker Peer-to-Peer SOA

2) 단계 2: 패턴의 우선순위 설정

분류한 패턴을 Table 7처럼 우선순위화하고 등장한 빈도수를 같이 작성한다. 이를 바탕으로 요구사항의 우선순위화 조건으로 추가해 우선순위를 결정할 수 있도록 하면 Table 8과 같이 정리된다. 외부 관계와 관련된 패턴과 매핑된 요구사항이 Priority 1로 가장 높고, 모듈 내부 관계와 관련된 패턴과 매핑된 요구사항이 Priority 2가 된다. 직접 구조화를 해야 하는 요구사항이 Priority 3으로 가장 낮은 우선순위를 가지게 된다. 각 패턴의 빈도수는 같은 우선순위인 경우에 적용 순서를 결정할 조건으로 사용한다.

Table 7. Prioritize Candidate Patterns

Priority	Criteria	Patterns
Priority 1	Patterns with external relationship	Peer-to-Peer(6) Client-Server(5) Broker(5) MVC(3), SOA(1)
Priority 2	Patterns with internal relationship	Layered(3) Publish-Subscribe(1)
Priority 3	Internal algorithm that is difficult to pattern	None(2)

Table 8. Prioritize Key Requirements of Sample Software

No.	Requirements	(Biz. value, Arch. impact)	Priority by PRT
1	uMediaServer should be able to deliver media commands (e.g. play, pause and etc.) to the video playback pipeline designated by the media application.	(H, M)	Priority 1
2	uMediaServer should be able to deliver commands (e.g. startRecord, stopRecord and etc.) to the camera pipeline specified by the camera application.	(H, M)	Priority 1
3	Pipelines are separate and shouldn't be called directly from the application.	(M, M)	Priority 2
4	uMediaServer must deliver notification events and errors related to playback.	(H, M)	Priority 1
5	uMultiple applications should be able to send play commands simultaneously.	(L, M)	Priority 1
6	uMediaServer should prioritize applications that can be played.	(M, H)	Priority 3
7	The communication system should be abstracted.	(L, M)	Priority 2
8	uMediaServer should always be running.	(M, M)	Priority 3
9	uMediaServer should provide applications with a separate communication library in the form of a proxy.	(L, H)	Priority 1
10	All command requests should be returned within 10msecs.	(M, H)	Priority 1

우선순위 기준에 따라 분류된 패턴들 중 빈도수가 높은 패턴을 선정하면 Priority 1 기준에서는 Peer-to-Peer 패턴이 선택되고, Priority 2 기준에서는 Layered 패턴이 선택된다. 이를 바탕으로 Table 8처럼 각 요구사항의 우선순위를 설정한다. Table 8의 3번 요구사항처럼 Priority 1 패턴과 Priority 2 패턴이 같이 관련된 경우 각 우선순위 조건에서 빈도수 순위에 따라 관련 패턴의 값을 설정한다. Layered 패턴이 Priority 2 조건에서 빈도수가 가장 많아 3번 요구사항은 Priority 2로 설정한다.

3) 단계 3: 외부 관계와 연관된 패턴 적용

외부 관계와 연관된 패턴을 적용한다. 이때 중복된 항목에서 Peer-to-Peer 패턴이 가장 높은 발생 빈도수를 보여 이를 적용한다. 기존에는 Broker 패턴 적용으로 Broker 역할을 했을 umediaserver를 제거하고 Peer-to-Peer 패턴을 이용하여 umedia-api와 resource_mgr_client가 직접 연결이 되도록 하여 Fig. 6과 같이 설계된다. Fig. 5와 큰 차이는 umediaserver 부분이 모두 제거되는 것이다.

4) 단계 4: 내부 관계와 연관된 패턴 적용

통신체계 분리를 위해 노드 별로 UMSConnector 클래스를 추가하여 Layer를 나누고 변경 용이성을 향상시키도록 한다. 이는 통신체계 부분을 분리하면서 서비스 로직 부분과 프로세스간 통신을 하는 부분을 분리하는 효과를 가지게 된다. Publish-Subscribe 패턴은 콜백 함수로 구현하여 별도로 나타나지 않는다. Fig. 7과 같이 설계된다. Fig. 6과 비교 시 umedia-api와 resource_mgr_client 내부에 UMSConnector가 추가된다.

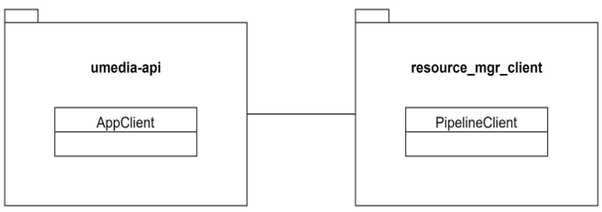

Fig. 6. 1st Architecture to Apply Patterns Related to External Relationship

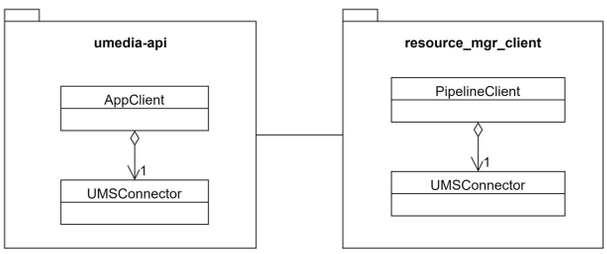

Fig. 7. 2nd Architecture to Apply Patterns Related to Internal Relationship

5) 단계 5: 내부 알고리즘 구조 적용

패턴을 지정하지 못한 요구사항들을 위해 구조를 추가한다. 애플리케이션들의 우선순위 정보를 위해 PipelinePriority를 추가한다. 브로커 역할을 하는 중앙 프로세스가 없기 때문에 특정 프로세스가 비정상 종료되어도 다른 프로세스의 동작에 영향을 주지 않아 기존에 발생할 수 있던 가용성 문제가 해결되며, 최종적으로 Fig. 8과 같이 설계된다.

기존 Fig. 5와 각 클래스 및 역할을 비교하면 Table 9와 같다. 브로커 역할이 없어지면서 중앙에서 정보를 수집하고 관리하던 uMediaServer와 하위 Controller들인 Pipeline Manager, ResourceManager, AcquireQueue, Pipeline Pool 역할들이 불필요해진다. 또한 모든 파이프라인의 상태 정보를 하는 Pipeline과 resource_request_t 정보 처리가 PipelineClient에서 각자의 정보를 관리하게 된다. 또한 다른 파이프라인들 사이에서 우선순위를 알아야 하기 때문에

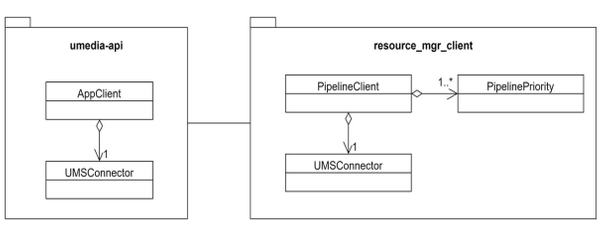

Fig. 8. 3rd Architecture with Internal Algorithm Applied

Table 9. Role Mapping in Original and Modified Umediaserver

No.	original umediaserver	modified umediaserver
1	uMediaServer	N/A
2	PipelineManager	N/A
3	Pipeline	PipelineClient
4	UMSConnector	UMSConnector
5	ProcessPool	N/A
6	Process	N/A
7	ResourceManager	N/A
8	resource_request_t	PipelineClient
9	AcquireQueue	N/A
10	acquire_request_connection_t	PipelinePriority
11	uMediaClient	AppClient
12	ResourceManagerClient	PipelineClient

PipelinePriority가 추가되어 분산된 환경에서 각자 우선순위를 확인하여 별도의 스케줄링 알고리즘도 불필요해진다. 하나의 인터페이스에 관련된 모든 클래스들이 AppClient 하나로 처리하고 파이프라인에 직접 연결되고 인터페이스를 호출하게 되어 전체 아키텍처가 Fig. 5보다 간결하게 설계된다.

6) 단계 6: 아키텍처 평가

최종 아키텍처를 변경 용이성과 성능 측면에서 평가해 본다. Peer-to-Peer 패턴 적용으로 인해 불필요한 브로커가 없어져 구조가 간단해진다. 미디어 인터페이스를 추가하거나 수정할 때 AppClient 클래스만 수정하면 되어 변경량이 기존 구조에 비해 적어지게 된다. 또한 우선순위 정책 변경 시 PipelinePolicy 클래스만 변경하면 된다. 추가로 resource_mgr_client의 상태 저장 부분을 모두 이관하면서 메모리 낭비가 없어지고, 미디어 명령 전달 시 서버를 거치지 않고, 버퍼링 과정이 제거되어 성능에 유리한 점이 있을 것으로 분석된다. 단점으로는 내부 알고리즘으로 처리하는 우선순위 관련 정보가 중앙에 집중된 구조가 아니라서 정보 동기화가 늦을 수도 있으나 기능은 정상 동작하고, 성능이 향상되는 부분이라 크게 문제 되지 않을 것으로 판단된다. 각 품질 속성의 평가지표로 평가해보면 Table 10과 같다. 변경 용이성(Modifiability)은 수정되는 클래스 수가 줄어들면서 0.15p 향상되고, 성능(Performance)은 버퍼링을 하는 process 수가 줄면서 0.16p 향상되었다.

Table 10. The Measurement Result of Original and Modified Design

Quality Attribute	Original	Modified	diff.
Modifiability	0.35 # = {3, 4}	0.5 # = {1, 1}	+0.15
Performance	0.34 # = {14, 1}	0.5 # = {5, 0}	+0.16

4.4 결과 고찰

경험적 직관에 의해 설계된 아키텍처와 PRT를 적용한 아키텍처를 비교할 때 PRT를 적용한 아키텍처가 변경 용이성 측면에서 더 효율적인 것으로 분석된다. 미디어 인터페이스 추가 시 기존 아키텍처는 4개의 클래스가 변경되어야 하지만 PRT를 적용한 아키텍처는 1개의 클래스가 변경되고, 스케줄링 정책 변경 시 수정되는 클래스가 3개에서 1개로 줄면서 0.15p 향상되었다. 또한 버퍼링 관련 프로세스가 제거되면서 성능이 0.16p 향상되었다. 추가로 PRT를 적용한 아키텍처에서는 브로커와 같은 복잡한 알고리즘의 프로세스가 없고, 관리를 위해 낭비하는 메모리가 상대적으로 적어 더 효율적인 아키텍처라고 판단된다. 이러한 연구 결과를 바탕으로 효율적인 설계를 체계적으로 할 수 있는 패턴 기반 요구사항 분류체계를 제안한다.

5. 결론 및 향후 연구

아키텍처 설계가 대부분 아키텍트의 직관에 의지하고 있다. 그럼에도 패턴에 관한 방법론 연구가 많이 이루어지고 있으나, 패턴의 적용 순서에 따른 효과성에 관해서는 연구된 바가 없다. 본 연구에서는 패턴의 적용 순서가 아키텍처 결과에 영향을 줄 수 있음을 보이고, 이를 기반으로 요구사항별로 패턴을 분류하고, 분류한 패턴들을 체계적으로 적용할 수 있는 패턴 기반 요구사항 분류체계인 PRT를 제안한다. 패턴의 적용 순서가 비즈니스 가치나 아키텍처 영향도보다 우선 고려될 수는 없지만, 같은 우선순위의 요구사항이나 반드시 반영해야 하는 리스트에 포함된 요구사항들의 우선순위화를 위해 본 논문에서 제안한 PRT가 도움이 될 수 있을 것으로 보인다.

향후 연구에서 PRT를 통한 우선순위화에 대해 더 효율적인 방안을 연구하고, 아키텍처 평가를 더 구체적으로 할 예정이다. 패턴의 적용 속성 키워드를 더 상세히 분류하게 되면 더 다양한 요구사항의 문구에 대응할 수 있을 것으로 보인다. 또한 사례연구에서 실험한 소프트웨어보다 요구사항이 많은 대규모 시스템에서 의존성이 있는 요구사항들을 그룹핑(grouping)하면서 패턴 유형과 품질 속성을 같이 고려하여 아키텍처를 설계해 PRT의 효용성에 대해 비교 적용해볼 수 있을 것으로 보인다. 또한 설계가 완료된 아키텍처를 평가할 때 ATAM(Architecture Trade-off Analysis Method) 방법론을 이용하여 더 구체적으로 아키텍처를 평가해 볼 수 있을 것이다.

References

[1] L. Bass, P. Clements, and R. Kazman, “Software Architecture in Practice,” Upper Saddle River (N.J.): Addison-Wesley, 2013.

[2] P. Clements, F. Bachman, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: Views and beyond," Addison-Wesley, 2002.
[3] K. Wiegers and J. Beatty, "Software requirements," Sydney: Pearson Education, 2013.
[4] M. Aychew and E. Alemneh. "Selection of architectural patterns based on tactics," *2022 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, IEEE, 2022.
[5] P. Kruchten, "Mommy, where do software architectures come from," In *Proceedings of the 1st Intl. Workshop on Architectures for Software Systems*, pp.198-205, 1995.
[6] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-oriented software architecture-A system of patterns," Chichester, U.K., John Wiley & Sons, 1996.
[7] M. Richards, "Software Architecture Patterns," 1st ed. O'Reilly, 2015.
[8] P. M. Jacob, and P. Mani, "Software architecture pattern selection model for Internet of Things based systems," *IET Software*, Vol.12, No.5, pp.390-396, 2018.
[9] M. Galster, A. Eberlein, and M. Moussavi, "Systematic selection of software architecture styles," Iet Software, Vol.4, No.5, pp.349-360, 2010.
[10] S. K. Lo, Q. Lu, H. Y. Paik, and L. Zhu. "A decision model for federated learning architecture pattern selection," *arXiv preprint arXiv:2204.13291*, 2022.
[11] M. Kassab, M. Mazzara, J. Lee, and G. Succi, "Software architectural patterns in practice: An empirical study," *Innovations in Systems and Software Engineering*, Vol.14, pp.263-271, 2018.
[12] D. K. Kim and C. El Khawand, "An approach to precisely specifying the problem domain of design patterns," *Journal of Visual Languages & Computing*, Vol.18, No.6, pp.560-591, 2007.
[13] J. Kim, S. Kang, and J. Lee, "A software architecture design method that matches problem frames and architectural patterns," *Journal of KIISE*, Vol.42, No.3, pp.341-360, 2015.
[14] D. Pandey, U. Suman, and A. K. Ramani, ''An effective requirement engineering process model for software development and requirements management,'' in *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, pp.287-291, 2010.
[15] Swit.or.kr, 2022. [internet] https://www.swit.or.kr/IS/web/isCbmRefView.jsp?ref_sq=1104&schCode=.
[16] E. Gamma, R. Johnson, R. Helm, R. E. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software," Pearson Deutschland GmbH, 1995.
[17] Github - webosose/umediaserver, Github, 2022. [Internet], https://github.com/webosose/umediaserver.

최 종 우

https://orcid.org/0009-0001-2718-7608
e-mail : jongwoos.choi@gmail.com
2010년 건국대학교 컴퓨터공학부(학사)
2023년 한국과학기술원 소프트웨어대학원 (석사)
2010년 ~ 2013년 삼성탈레스 해양/시스템연구소 연구원
2013년 ~ 2015년 LG전자 SW플랫폼연구소 주임연구원
2015년 ~ 현 재 LG전자 HE연구소 책임연구원
관심분야 : 소프트웨어 공학, 소프트웨어 아키텍처, 소프트웨어 신뢰성, 인공지능

민 상 윤

https://orcid.org/0009-0002-4107-368X
e-mail : sang@sol-link.com
1993년 San Francisco State University 전산학과(학사)
1997년 한국과학기술원 정보통신공학과 (석사)
2003년 한국과학기술원 전산학과(박사)
2000년 ~ 현 재 (주)솔루션링크 대표이사
2020년 ~ 현 재 (사)소프트웨어와사회안전협회 협회장
2004년 ~ 현 재 한국과학기술원 전산학부 겸직교수
2020년 ~ 현 재 IEEE P1228 표준화 위원
2017년 국회 소프트웨어 안전 포럼 추진위원장/부의장
2012년 ~ 2015년 소프트웨어안전 전문가포럼 운영위원장
관심분야 : 소프트웨어 공학, 소프트웨어 안전, 안전 법・제도