프로덕트라인 아키텍처의 정량성 평가 기법
(A Metric-based Methodfor Evaluating Product Line Architecture)

장 수 호 † 라 현 정 † 김 수 동 ††
(Soo Ho Chang) (Hyun Jung La) (Soo Dong Kim)

요 약 프로덕트라인 공학(Product Line Engineering, PLE)은 여러 어플리케이션들이 공유할 수 있는 핵심자산을 사용하는 대표적인 재사용 방법이다. 프로덕트라인 아키텍처(Product Line Architecture, PLA)는 핵심자산의 주요 구성 요소 중 하나이다. PLA는 일반적인 소프트웨어 아키텍처와는 달리 한 프로덕트라인에 속한 여러 멤버의 공통성과 가변성을 포함하므로, 기존 아키텍처와는 다른 방법으로 평가되어야 한다. 그러나, 기존 연구는 PLA와 기존 아키텍처와의 차이를 충분히 다루고 있지 못하여 PLA 평가는 PLE에서 어려운 작업 중의 하나로 인식된다. 본 논문에서는 PLA 설계 시에 중요하게 다루어져야 하는 두 가지 이슈를 제안한 후 식별한 두 가지 이슈를 중점으로 PLA를 평가하기 위한 메트릭을 제안한다. 식별된 두 가지 이슈와 제안된 PLA 평가 메트릭으로 PLA를 효율적으로 설계할 수 있으며, 제안된 메트릭으로 아키텍처 설계자의 경험에 의해 수행되는 PLA 평가를 좀 더 체계적으로 수행할 수 있다.

키워드 : 프로덕트라인 공학, 아키텍처, 아키텍처 평가기법, 메트릭

Abstract Product Line Engineering (PLE) has been widely accepted as a representative software reuse methodology by using core assets. As a key element of core assets, product line architecture (PLA) should be generic to a set of applications in the product line (PL). However, the difference between PLA and single system architecture has not been treated well enough, so evaluating PLA still remains as one of the difficult tasks in PLE. In this paper, we identify two intrinsic but overlooked issues in PLA; *variability propagation chain and conflicts between architectural elements*. And, we present a metric-based methodto evaluate PLA from the perspective of the two issues. We believe that the two issues in PLA and the evaluation method would make designing high-quality PLA more feasible and effective.

Key words : Product Line Engineering, Architecture, Architecture Evaluation, Metric

1. 서 론

프로덕트라인 공학(Product Line Engineering, PLE)은 여러 어플리케이션들이 공유할 수 있는 핵심자산을 사용하는 대표적인 소프트웨어 재사용 방법이다. PLE는 크게 핵심 자산 공학 단계(Core Asset Engineering)와 어플리케이션 공학 단계(Application Engineering)로 분류된다. 핵심 자산 공학 단계에서는 여러 프로덕트라인(Product Line, PL) 멤버들이 공통으로 재사용할 수 있는 핵심 자산을 식별하고 개발하며, 어플리케이션 공학 단계에서는 미리 정의된 핵심 자산을 목적에 맞게 인스턴시에이션(Instantiation)하여목표 어플리케이션을 개발한다. 핵심 자산은 하나의 PL에 속하는 패밀리 멤버들이 어플리케이션을 만드는데 기초가 되는 모든 자산을 의미하며, PLA, 컴포넌트, 결정 모델 등이 있다. 이 중 프로덕트라인 아키텍처(Product Line Architecture, PLA)는 PL 멤버들이 사용하는 어플리케이션의 전반적인 구조를 나타내며, PL의 범위를 정하는데도 도움이 되는 핵심 요소이다[1,2]. PLA는 하나의 어플리케이션을 위한 아키텍처와는 달리 하나의 PL에 속하는 여러 어플리케이션을 고려해야 한다. 그러므로, PLA을 평가하기 위하여 아키텍처 가변성과 PL에 속하는 여러 어플

· 본 연구는 한국과학재단 특정기초연구(R01-2005-000-11215-0)지원으로 수행되었음
† 학생회원 : 숭실대학교 컴퓨터학과
shchang@otlab.ssu.ac.kr
hjla@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
논문접수 : 2005년 11월 4일
심사완료 : 2006년 3월 16일

리케이션들의 다양한 비기능적 요구사항에 중점을 두어야 한다.

PLE에서 가변성은 핵심자산의 재사용성에 영향을 주는데 이러한 가변성은 PLA 수준에서도 존재한다. PLA의 가변성 설계의 평가는 아직 해결되지 않은 많은 연구항목을 가지고 있어 지속적인 연구가 필요하다. PLA 평가는 비기능적 요구사항을 다양한 어플리케이션의 집합 관점에서 설계된 것으로 평가해야 하므로 더 많은 전문적인 지식이 필요하며 더 복잡할 수 있다[3]. 이러한 PLA의 특징으로 '가변성 파급에 대한 이슈'와 '아키텍처 요소 간의 충돌에 대한 이슈'를 도출된다.

본 논문에서는 PLA 평가를 보다 체계적으로 접근하기 위해 2장에서 PLA와 관련된 연구에서 제안된 평가 방법을 알아본다. 그리고, 3장에서는 PLA 설계시 고려해야 할 이슈들을 식별하고, 4장에서는 이슈를 기반으로 PLA를 평가하기 위한 메트릭을 제안한다. 그리고, 5장에서는 사례연구를 통해 본 논문에서 정의된 메트릭이 어떻게 적용되는지에 살펴본다. 식별된 두 가지 이슈와 이를 기반으로 제안된 PLA 평가 메트릭은 높은 품질의 PLA를 설계하는데 도움이 되며, 아키텍처 설계자의 경험에 의해 수행되는 PLA 평가를 좀 더 체계적으로 수행할 수 있게 된다.

2. 연구 배경

2.1 관련 연구

Leire와 Goiuria의 연구에서는 프로덕트라인 아키텍처와 연관된 속성을 세가지로 구분하였다[4]. 프로덕트라인 품질속성은 프로덕트라인과 관련된 제품 및 앞으로 사용될 가능성이 있는 제품들에 대한 기반 내용을 가지며, 수정가능성과 구성가능성으로 구분되는 가변성과 관련된 내용들이 속한다. 도메인 관련 품질속성은 특정한 도메인에 대한 주요 품질 속성을 정의하며 성능, 안정성, 보안성 등의 품질 속성이 이에 속한다. 일반적인 기능은 프로덕트라인의 멤버들이 공통으로 가지는 기능에 관한 내용을 포함한다. 이러한 세가지 속성을 기반으로 각 연관 속성에서 활용될 수 있는 기존의 소프트웨어 아키텍처 평가기법을 조사하여 소개하였다. 이 소개에는 각 평가기법들이 표의 형태로 정리되어 있다. 그러나 각 연관 속성에 대한 기존의 소프트웨어 아키텍처 평가기법의 소개가 목적이어서, PLA 평가에 적용하기 위한 지침과 별도의 평가 방법은 소개되지 않는다.

HoPLAA에서는 기존 소프트웨어 아키텍처의 대표적인 평가 방법 중 하나인 ATAM을 확장하여, 가변점을 정성적으로 분석하고 PLA를 PL 멤버들이 원하는 다양한 품질 시나리오 간의 우선순위를 고려하여 평가 한다[5]. 이 방법에서는 크게 PLA를 평가하는 단계와 인스턴시에이트하여 만든 프로덕트 아키텍처를 평가하는 단계로 분류하고, 각 단계는 7개의 절차로 구성된다. PLA를 평가하는 첫 번째 단계에서는 가변점을 평가하여 민감한 지점(sensitivity point)과 교환(tradeoff) 지점 외에 진화 가능한 지점(evolvabiltiy point)과 진화를 위한 제약사항(evolvability constraint)을 생성하고, 프로덕트 아키텍처를 평가하는 두 번째 단계에서는 PLA에 명시된 품질 요구사항이 프로덕트 아키텍처에 유효하게 사용될 수 있는지, PLA가 프로덕트에만 존재하는 품질 요구사항을 지원할 수 있는지를 평가하여 기존 ATAM과 같은 산출물을 만든다. 진화 가능한지점과 진화를 위한 제약사항은 프로덕트 아키텍처를 인스턴시에이션 할 때 프로덕트에만 적용되는 품질 요구사항을 지원하기 위한 것이다. 그러나, 이 논문에서 제시한 방법은 PLA 평가를 정량적으로 접근하기 보다는 아키텍처 설계자의 경험과 지식에 의해 다르게 식별되는 시나리오 기반으로 PLA를 평가하므로 체계적이며 정량적인 평가 결과값을 얻을 수 있는 접근 방법이 필요하다.

Niemelia는 아키텍처를 평가하기 위해 아키텍처만 고려하지 않고, 아키텍처를 비즈니스, 프로세스, 기관을 연관시켜 평가한다[6]. 이 논문에서 제시한 아키텍처 평가 방법은 총 네 가지 단계로 이루어진다. 첫째, 프로덕트 패밀리 아키텍처를 위한 평가 프레임워크를 설문지 형식으로 만든다. 이 평가 프레임워크는 비즈니스, 아키텍처, 프로세스, 기관을 모두 고려하여 작성된다. 두번째 단계에서는 정의된 평가 프레임워크를 기준으로 아키텍처 명세서를 검토한다. 다음으로 평가 결과를 분석한 뒤, 마지막으로 아키텍처의 성숙도를 네 가지 측면(프로덕트 패밀리 아키텍처 측면, 프로덕트 품질 측면, 재사용성 측면, 도메인 측면)에 맞춰 분류한다. 이 측면은 각각 성숙도에 따라 5가지의 단계로 분류된다. 이 논문에서 제시한 평가 프레임워크는 대개 설문지 형식이기 때문에 다소 평가자의 주관에 의해 정성적으로 아키텍처를 평가하게 되므로, 평가자의 주관이 아닌 보다 정량적이고 체계적으로 평가하는 접근 방법이 필요하다.

2.2 기반 연구

이번 절에서는 본 논문에서 제시하는 PLA 평가 대상이 되는 아키텍처 요소를 기존 아키텍처 연구와 PLA 연구를 기반으로 도출한다.

PLA에 대한 여러 연구에서는 아키텍처 구성 요소를 다음과 같이 기술한다. PuLSE 방법론[7]에서는 구체적으로 컴포넌트와 컴포넌트 간의 관계를 명시하고 있지 않으나 명세(모델)에 그 내용이 내포되어 있고, 다른 방법론과 달리 가변성 모델을 아키텍처의 구성요소로 구

분한다. Bosch[1]의 PLA는 시스템과 시스템의 환경을 인터페이스로정의한 참조 컨텍스트 외에 아키텍처를 분해하는 과정에서 생성된 컴포넌트와 컴포넌트간 관계를 표현하는 '구조'(Structure)를 아키텍처의 구성요소로 기술하였다. COPA[8] 와 QADA[9]에서는 아키텍처의 구성요소를 컴포넌트와 컴포넌트간 관계로만 명시하는 것 외에 아키텍처뷰가 고려되었다. KobrA[10]에서는 아키텍처가 구체적으로 명시되지 않지만, 컨테인먼트트리와 컨텍스트 실현으로 유추할 수 있고, 구성요소는 컴포넌트와 컴포넌트 간 관계로 이루어진다. 그러나, 일부 PLE 방법론의 PLA는 SEI에서 제시한 아키텍처나 IEEE, P1471 아키텍처에서 중요시하는 아키텍처뷰타입과 스타일[6,11]을 구성요소에 포함시키지 않는다. 사용자의 기능적인 요구사항과 비기능적인 요구사항(품질 요소)을 효과적으로 설계하기 위해 소프트웨어 아키텍처는 뷰와 스타일을 사용한다.

기존 프로덕트라인 공학에서 제시된 PLA 구성요소의 구성요소와 소프트웨어 아키텍처의 구성요소의 비교 결과로부터 본 논문에서는 PLA의 구성요소를 아키텍처 스타일, 컴포넌트와 컴포넌트 간의 관계, 가변성 정보를 포함하는 아키텍처 결정 모델로 구분한다.

- **아키텍처 스타일(Style)**은 컴포넌트와 컴포넌트 간 관계로 특정 문제를 만족시키고 해결하기 위해 미리 설계된 패턴이다. 아키텍처는 소프트웨어를 바라보는 적절한 뷰를 선택하고, 그 뷰에 속하는 스타일을 아키텍처 드라이버에 맞춰 선택한 후, 스타일에 맞게 컴포넌트와 컴포넌트 간관계를 추출하는 과정을 거쳐 설계된다[12]. PLE에서는 여러 PL 멤버를 위한 아키텍처 드라이버를 만족시키기 위해 다양한 아키텍처 스타일을 선택한다. 따라서, 드라이버와 마찬가지로 가변적인 스타일이 존재할 수 있다.
- **컴포넌트(Component)와 컴포넌트 간 관계(Inter-component Relationship)**는 아키텍처를 구성하는 요소이다. 컴포넌트는 아키텍처의 기능적 요구사항과 비기능적 요구사항을 구현한 단위로, 데이타와 기능을 가지는 객체(Object)와 객체간의 관계로 구성되어 있다. 각 컴포넌트는 인터페이스(Interface)를 통해 외부로 기능이 보여지며, 컴포넌트의 기능을 사용하기 위해서는 인터페이스를 통해야 한다. 컴포넌트 간 관계는 컴포넌트가 고유한 기능을 완벽하게 수행하기 위해 필요한요소로, 하나의 컴포넌트는 스스로 고유한 기능을 수행하지 못하는 경우에 다른 컴포넌트와 관계를 맺는다.
- **아키텍처 결정 모델(Decision Model)**에는 프로덕트라인에속한 여러 패밀리 멤버에 재사용되기 위한 가변적인 아키텍처 요소들에 모든 정보를 포함하며, 결정 모델에 포함되는 가변성은 아키텍처 드라이버에 대한 가변성, 아키텍처 스타일에 대한 가변성, 컴포넌트와 컴포넌트간 관계에 대한 가변성이 있다. 소프트웨어 아키텍처가 범용 아키텍처와 크게 다른 점은 바로 범용 아키텍처가 결정 모델을 포함한다는 것이다.

위에서 알아본 요소들은 PLA에 설계에 참조되거나 PLA를 구성하는 요소로, 주요한 평가 대상이 되지만, 기존 소프트웨어 아키텍처에서는 이 외에도 아키텍처 설계를 유도하는 요구사항인 아키텍처 드라이버를 주요한 평가 대상으로 취급한다[6]. 아키텍처 드라이버(Architectural Driver)는 어플리케이션의 아키텍처 설계에 중요하게 영향을 주는 요구사항을 의미하며[12], 기능적 요구사항, 품질 요구사항, 비즈니스 요구사항이 포함된다. 예를 들어, '보안성', '사용성' 등이 이에 포함된다. 본 논문에서는 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트와 컴포넌트 간 관계를 합쳐서 아키텍처 요소라 한다.

3. PLA 설계의 특징적 이슈

PLE에서는 PL 내에 포함된 멤버들의 공통성 정보뿐만 아니라 가변성 정보는 반드시 PLA 설계에 포함되어야 한다. 여러 멤버들의 공통성과 가변성 정보를 다루고 있을 뿐 아니라 PL 멤버들이 공용할 수 있는 PLA를 설계해야 하기 때문에 기존의 하나의 어플리케이션을 위한 아키텍처 설계 과정에서 나타나지 않았던 새롭고 복잡한 이슈들이 도출된다. 이런 이슈들은 PLA 설계에서 중요하게 다루어져야 하지만, 기존 PLA 연구에서는 다소 깊게 다루고 있지 않는다. 이번 장에서는 PLA 설계 과정에서 나타나는 '가변성 파급 효과에 대한 이슈'와 '아키텍처 요소 간의 충돌 이슈'에 대해 알아본다.

3.1 이슈 1: 가변성의 파급

아키텍처 드라이버의 가변성: 정의에 의해, PL은 핵심 자산을 공유하는 여러 프로덕트인 P_1, P_2, …, P_n로 구성되고, 각 P_i는 아키텍처 드라이버 집합을 가진다. 즉, P_i의 아키텍처 드라이버 집합인 $SetD_i$를 다음과 같이 정의한다.

$SetD_i$ = {D_{ij} D_{ij} 는 i번째 프로덕트의 j번째 드라이버이다.}

일반적으로 PL 요구사항은 PL멤버가 공통으로 사용할 수 있는 휘처를 포함하지만, 가변적인 휘처도 포함한다. 그러므로, 두 프로덕트인 P_i 와 P_j는 다른 아키텍처 요구사항을 가지게 되고, 두 프로덕트의 아키텍처 드라이버 집합인 $SetD_i$ 와 $SetD_j$는 같지 않게 된다. 이런 차이점을 아키텍처 드라이버의 가변성이라고 부르며, 이는 PLA 설계시 적절하게 고려되어야 한다.

일부 아키텍처 드라이버는 여러 프로덕트에서 공통적

으로 포함되는 필수 드라이버인 반면에, 다른 아키텍처 드라이버는 여러 프로덕트에 따라 다르게 적용되는 선택 드라이버 또는 대안 드라이버이다. 이에 따라 분류한 아키텍처 드라이버를 다음과 같은 용어로 정의한다.

- *MandatorySetD*는 한 PL에 속하는 멤버가 공통으로 사용하는 아키텍처 드라이버의 집합이다.
- *AlternativeSetD*는 추상적인 상위 수준의 드라이버는 공통적으로 존재하지만, 상세 수준의 드라이버들이 조금씩 다르게 정의되는 아키텍처 드라이버들의 집합이다. 예를 들어, 한 멤버는 어플리케이션에서 발생하는 모든 서비스의 로그를 기록하여 안정성을 유지하지만 다른 멤버는 방화벽을 사용하여 외부 침입으로부터 어플리케이션을 보호하길 원한다고 하자. 이 경우 두 멤버는 방화벽과 로깅과 같은 상세수준의 드라이버는 다르지만 상위수준의 드라이버는 보안성으로 공통이다.
- *OptionalSetD*는 PL에 속하는 모든 멤버에서 사용되지 않고, 일부 멤버에 의해서만 사용되는 아키텍처 드라이버들의 집합이다.

그러므로, PL에 포함되는 아키텍처 드라이버들의 집합인 *PLSetD*은 *MandatorySetD*, *AlternativeSetD*, *OptionalSetD*로 구성되고 다음과 같이 식으로 표현할 수 있다.

PLSetD = {MandatorySetD, AlternativeSetD, OptionalSetD}

아키텍처 스타일의 가변성: 아키텍처 드라이버는 아키텍처 스타일을 선택할 수 있는 기준이 되며, 적절한 스타일이 선택됨으로써 해당 아키텍처 드라이버가 PLA 설계에 반영된다. 예를 들어, 아키텍처 드라이버인 확장성은 계층 스타일이나 피어 투 피어(peer-to-peer) 스타일을 선택함으로써 아키텍처 설계에 반영된다. 만약 아키텍처 드라이버에 가변성이 존재한다면, 이를 설계에 반영하기 위해 선택된 아키텍처 스타일이나 패턴에도 가변성이 존재할 수 있다.

$SetS_i$ 는 아키텍처 드라이버 집합인 $SetD_i$를 만족시키는 스타일의 집합인 아키텍처 스타일들의 집합이고, 다음과 같이 정의한다

$SetS_i$ = {S_{ij} S_{ij} 는 *i번째 프로덕트의 j번째 스타일이다.*}

집합 $SetS_i$의 모든 스타일에도 아키텍처 드라이버와 같이 필수 아키텍처 스타일, 선택 아키텍처 스타일, 대안 아키텍처 스타일로 분류된다. 대개 *MandatorySetD*에 속한 드라이버는 하나 이상의 필수 아키텍처 스타일을 유도하고, 마찬가지로 *AlternativeSetD*와 *OptionalSetD*에 속한 드라이버는 각각 하나 이상의 대안 아키텍처 스타일과 선택 아키텍처 스타일을 유도한다. 이제 가변성 종류에 따라 분류한 아키텍처 스타일을 다음과 같이 정의한다.

- *MandatorySetS* 는 한 PL에 속하는 모든 멤버들이 공통으로 사용하는 아키텍처 스타일들의 집합이다.
- *AlternativeSetS* 는 *AlternativeSetD*에 있는 대안 아키텍처 드라이버에서 유도된 스타일들의 집합이다.
- *OptionalSetS* 는 PL에 속하는 모든 멤버에서 사용되지 않고, 일부 멤버에 의해서만 사용되는 아키텍처 스타일들의 집합이다.

그러므로, PL에 포함되는 아키텍처 스타일들의 집합인 *PLSetS*은 *MandatorySetS*, *AlternativeSetS*, *OptionalSetS*로 구성되고 다음과 같이 식으로 표현할 수 있다.

PLSetS = {MandatorySetS, AlternativeSetS, OptionalSetS}

컴포넌트와 컴포넌트 간 관계의 가변성: 기능적 요구사항은 컴포넌트와 컴포넌트 간 관계로 실현되고, 비기능적 요구사항은 컴포넌트와 컴포넌트 간 관계로 이루어진 PLA의 전반적인 구조를 정의하는데 중요한 역할을 한다. 컴포넌트와 컴포넌트 간 관계는 기능적 요구사항뿐 아니라 비기능적 요구사항의 가변성에 영향을 받기 때문에 PLA에 한 컴포넌트나 컴포넌트 간 관계가 다양한 형태로 존재할 수 있다. 이제 가변성종류에 따라 컴포넌트를 다음과 같이 정의한다.

- *MandatorySetCom* 은 한 PL에 속하는 모든 멤버들이 공통으로 사용하는 컴포넌트들의 집합이다
- *OptionalSetCom* 은 모든 PL 멤버가 아닌 일부 멤버에 의해 사용되는 컴포넌트들의 집합이다.
- *AlternativeSetCom* 은 한 요구사항을 만족시키기 위해 제공되는 여러 선택 가능한 컴포넌트 중에서 한 컴포넌트를 사용해야 하는 컴포넌트들의 집합이다.

컴포넌트와 마찬가지로, 컴포넌트 간 관계도 가변성에 따라 *MandatorySetComRel*, *OptionalSetComRel*, *AlternativeSetComRel* 로 분류된다.

그러므로, PLA를 이루고 있는 컴포넌트와 컴포넌트 간 관계의 집합을 나타내는 *PLSetCom* 은 다음과 같이 표현된다. 가변성에 따라 이렇게 분류된 컴포넌트와 컴포넌트 간 관계는 PLA 설계 시에 고려되어야 한다.

PLSetCom = { MandatorySetCom, AlternativeSetCom, OptionalSetCom, MandatorySetComRel, AlternativeSetComRel, OptionalSetComRel }

가변성 파급 체인: PLA 설계시 PLSetD 를 만족시키기 위해 PLSetS 를 선택하기 때문에, 가변적인 아키텍처 드라이버는 가변적인 아키텍처 스타일로 파급된다. 이와 유사하게, 가변적인 아키텍처 드라이버나 가변적인 아키텍처 스타일은 가변적인 컴포넌트와 컴포넌트 간 관계를 유도한다. 이러한 가변적인 요소들의 유도 관계를 가변성 파급 체인(Variability Propagation Chain)이

라고 정의한다. 그림 1은 가변적인 요구사항에서 컴포넌트와 컴포넌트 간 관계까지의 가변성 파급 체인을 나타낸다.

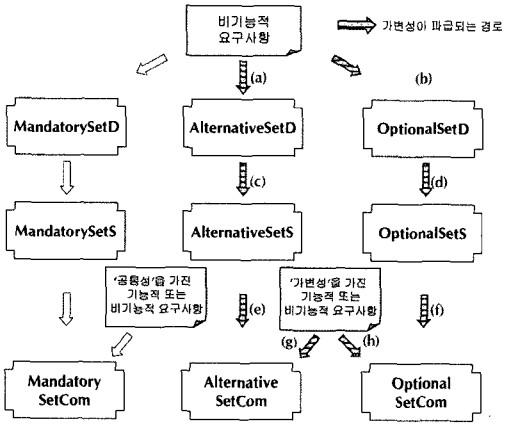

그림 1 아키텍처 가변성과 이들의 가변성 파급 체인

(a) →(c) →(e): (a)에서는 비기능적 요구사항에 포함된 가변성이 *AlternativeSetD*에 반영되어야 하며, *AlternativeSetD*는 (c)를 따라 *AlternativeSetS*를 유도한다. 그리고, (e)와 같이 *AlternativeSetS*를 실현하기 위해 가변적인 컴포넌트와 컴포넌트 간 관계로의 구성이 필요하다. (b) →(d) →(f)는 대안 가변성이 아니라 선택 가변성에 초점을 둔다는 점을 제외하고 앞의 경우와 유사하다. (g)와 (h)는 기능적 또는 비기능적 요구사항이 컴포넌트와 컴포넌트간 관계의 가변성에 영향을 줄 수 있음을 나타낸다.

3.2 이슈 2: 아키텍처 요소 간의 충돌

PLA는 여러 PL 멤버의 다양한 요구사항을 반영한 범용 아키텍처이다. 그러므로, 아키텍처 요소 사이에는 기존의 아키텍처에서 보다 복잡한 경우가 발생할 수 있으며, 그 대표적인 경우로 아키텍처 요소 간의 '충돌'이 있다.

아키텍처에서 '충돌'이란 같이 존재할 수 없거나 서로 대조되는 아키텍처 요소들이 서로 함께 존재해야 하는 관계를 가지고 부조화를 이루는 상태를 말한다[1]. PLA를 설계하고 PLA를 하나의 프로덕트 아키텍처로 인스턴시에이션하는 과정에서 아키텍처 요소 간의 충돌은 여러 시점에서 발생할 수 있다. 그림 2에서 충돌이 발생할 수 있는 시점을 '✖'로 표시하였다.

(a)로 표시된 첫 번째 경우의 충돌은 아키텍처 드라이버를 정의하는 과정에서 발생하는데, PL 멤버들 사이에 서로 다른 요구사항이 서로 충돌을 일으킨다. 만약 아키텍처 드라이버가 서로 충돌 관계에 있다면, 이로부터 유도된 스타일들의 충돌이 발생할 수 있다. 두 번째

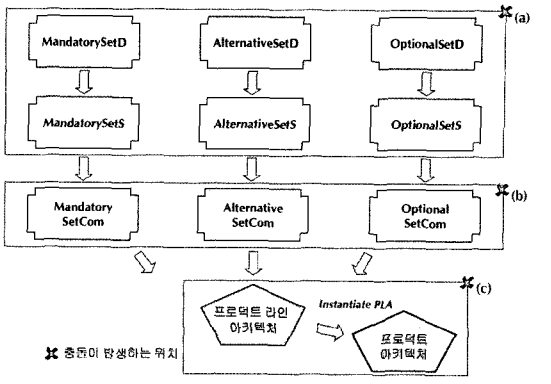

그림 2 아키텍처 요소 간 충돌이 발생하는 위치

경우는 (b)로 표시된 시점이며, 컴포넌트와 컴포넌트 간 관계를 정의하는 과정에서 충돌이 발생한다. 컴포넌트와 컴포넌트 간 관계의 충돌은 비기능적 요구사항뿐 아니라 기능적인 요구사항의 충돌이 원인이 된다. (c)로 표기된 세 번째 경우는 PLA를 하나의 프로덕트 아키텍처로 인스턴시에이션 하는 과정에 충돌이 발생하는 경우이다. PL 멤버는 PLA 요구사항과 상충되는 해당 프로덕트에만 적용되는 요구사항을 가지고 있을 수 있기 때문이다. PLA 요구사항과 프로덕트 요구사항이 충돌을 일으킬 경우에는 각 두 요구사항에서 유도된 아키텍처 스타일과 컴포넌트 간의 충돌도 초래한다.

아키텍처 요소 간의 충돌은 충돌을 일으키는 아키텍처 요소의 가변성 종류에 따라 그림 3과 같이 세 가지 타입으로 분류된다

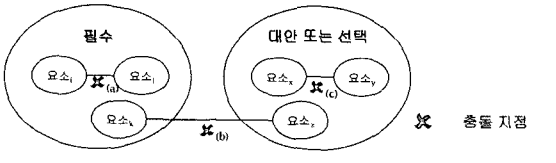

그림 3 아키텍처 요소 간 충돌의 여러 타입들

위의 세 가지 타입의 충돌에서 충돌을 일으키는 아키텍처 요소는 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트와 컴포넌트 간 관계로 세분화될 수 있다.

- (a) – 필수 아키텍처 요소들 사이의 충돌을 나타내며, 이는 한 시스템을 위한 아키텍처에서 발생하는 충돌과 같고, 해결하는 방법도 한 어플리케이션을 위한 아키텍처와 같다
- (b) – 필수 아키텍처 요소와 대안 또는 선택 아키텍처 요소 사이의 충돌을 나타낸다. 만약 목표 어플리케이션에서 가변적인 아키텍처 요소가 선택되지 않는다면, 이 형태의 충돌은 무시될 수 있다. 그러나, 가변적

인 아키텍처 요소가 선택된다면, (a)의 경우로 적용되어 해결된다.

- (c) - 대안 또는 선택 아키텍처 요소들 사이의 충돌을 나타낸다. 두 요소가 다른 목표 어플리케이션에서 사용된다면 무시될 수 있지만, 한 목표 어플리케이션에서 사용된다면 (a)의 충돌 해결 방법이 적용되어 PLA가 설계되고 인스턴시에이션이 수행되어야 한다.

위에서 알아본 '가변성 파급 효과에 대한 이슈'와 '아키텍처 요소 간의 충돌 이슈'는 PLA 설계뿐 아니라 설계된 PLA를 평가할 때 고려되어야 한다.

4. PLA 평가 메트릭

이번 장에서는 PLA를 평가하기 위한 세 가지 메트릭인 아키텍처 요구사항에 대한 준수성(Architectural Requirement Conformance, ARC), 아키텍처 충돌에 대한 안정성(Free of Conflict, FoC), 특화 가능성(Tailability, TAL)을 제안한다. 세 가지 메트릭은 3장에 기술한 PLA를 이루는 요소와 4장에서 식별된 두 가지 이슈를 기반으로 도출한 것이다. 그림 4는 평가 메트릭과 PLA 구성요소, PLA 이슈간의 관계를 보여준다.

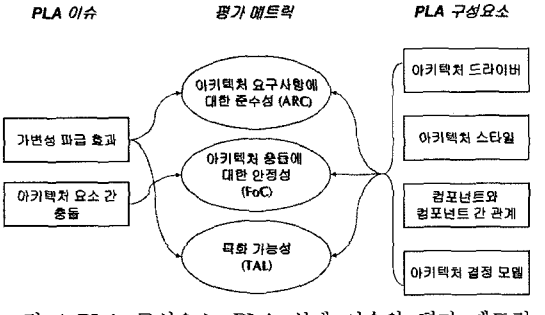

그림 4 PLA 구성요소, PLA 설계 이슈와 평가 메트릭 간의 관계

ARC 메트릭은 아키텍처 요구사항을 얼마나 만족하는지 측정하는 것으로, 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트와 컴포넌트 간 관계를 고려하며 이는 가변성 파급 효과로부터 도출된다. 그리고, FoC 메트릭은 PLA가 아키텍처 충돌에 얼마나 안정적인지를 측정하는 것으로, 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트의 충돌 개수를 고려하며, 아키텍처 요소 간 충돌 이슈에 초점을 두어 도출된 메트릭이다. 마지막으로, TAL 메트릭은 PLA가 목표 어플리케이션에 맞게 특화될 수 있는지를 측정하는 것으로, 아키텍처 결정 모델에 포함된 가변적인 드라이버, 스타일, 컴포넌트, 컴포넌트 간 관계를 이용하며, 이는 가변성 파급 효과 이슈에 초점을 두어 PLA를 평가한다.

4.1 아키텍처 요구사항에 대한 준수성

ARC 메트릭은 PLA가 얼마나 완전하게 아키텍처 요구사항을 만족하는지 측정하기 위한 메트릭이다. 아키텍처 드라이버는 아키텍처 요구사항을 분석함으로써 식별된다. 요구사항 명세서는 텍스트 형태로 작성되는 반면아키텍처 드라이버는 정형화된 항목으로 분류된다. 한번 아키텍처 드라이버를 식별되면, 이후의 PLA 설계 작업에서는 아키텍처 요구사항이 아닌 아키텍처 드라이버를 참고하여 설계한다. 따라서, ARC는 PLA가 얼마나 많은 아키텍처 드라이버를 만족하는지를 분석함으로써 측정된다.

ARC는 필수 아키텍처 드라이버에 일치하는 정도(Degree of Mandatory Driver Conformance, DMDC), 선택 아키텍처 드라이버에 일치하는 정도(Degree of Optional Driver Conformance, DODC), 대안 아키텍처 드라이버에 일치하는 정도(Degree of Alternative Driver Conformance, DADC)의 세 가지 세부 메트릭을 이용하여 측정된다.

$$DMDC = \frac{(n(SetD_{MandatorySetS} \cup SetD_{MandatorySetCom}))}{n(MandatorySetD)}$$

DMDC는 PLA가 얼마나 많은 필수 아키텍처 드라이버를 만족하는지를 측정하기 위한 메트릭이다. 위의 식에서, $SetD_{MandatorySetS}$와 $SetD_{MandatorySetCom}$는 각각 필수 아키텍처 스타일의 집합과 필수 컴포넌트 집합을 유도하는 필수 아키텍처 드라이버들의 개수를 나타낸다. 즉, DMDC는 *MandatorySetS*와 *MandatorySetCom*을 유도하는 필수 아키텍처 드라이버의 수를 한 PL에 있는 필수 아키텍처 드라이버의 집합인 *MandatorySetD*의 수로 나누어서 얻어진다.

DODC와 DADC는 각각 PLA가 얼마나 많은 선택 아키텍처 드라이버와 대안 아키텍처 드라이버를 만족하는지 측정하는 값으로, 필수 아키텍처 요소가 아닌 선택 또는 대안 아키텍처 요소에 초점을 둔다는 점을 제외하고 DMDC와 유사한 방식으로 계산된다.

$$DODC = \frac{(n(SetD_{OptionalSetS} \cup SetD_{OptionalSetCom}))}{n(OptionalSetD)}$$

$$DADC = \frac{(n(SetD_{AlternativeSetS} \cup SetD_{AlternativeSetCom}))}{n(AlternativeD)}$$

ARC는 다음에 나온 식과 같이 세 개의 세부 메트릭(DMDC, DODC, DADC)에 대한 평균값을 계산하여 얻어진다.

$$ARC = (DMDC + DODC + DADC) / 3$$

위의 공식으로 계산된 ARC 값의 범위는 0과 1사이가 된다. ARC 값이 0에 가까울수록 PLA는 아키텍처 요구사항의 많은 부분을 반영하지 않은 것을 의미하고, ARC 값이 1에 가까울수록 PLA는 아키텍처 요구사항의 대부분을 반영함을 의미한다.

4.2 아키텍처 충돌에 대한 안정성(Free of Conflicts, FoC)

FoC 메트릭은 PLA를 구성하는 아키텍처 요소들이 아키텍처 충돌을 발생시키지 않고 얼마나 안정적인지를 측정한다. FoC는 아키텍처 드라이버 충돌에 대한 안정성(Free of Driver-Conflict, FoDC)과 컴포넌트 충돌에 대한 안정성(Free of Component-Conflict, FoCC)의 두 가지 서브 메트릭을 이용하여 측정한다.

$$FoDC = 1 - (\frac{\text{드라이버 충돌 개수}}{n(PLSetD)C_2})$$

FoDC는 PLA가 얼마나 아키텍처 드라이버 충돌을 발생시키지 않는지를 측정하는 값으로, 이는 1에서 아키텍처 드라이버 충돌이 발생할 수 있는 정도를 뺀 값이다. 아키텍처 드라이버 충돌이 발생할 수 있는 정도는 아키텍처 드라이버 충돌이 발생하는 수를 PLSetD에 있는 아키텍처 드라이버간의 연관 가능한 모든 경우의 수를 나누어 구한다. 아키텍처 드라이버 간의 모든 연관 가능한경우의 수는 $_{n(PLSetD)}C_2$와 같이 조합을 이용하여 구한다.

$$FoCC = 1 - (\frac{\text{컴포넌트 충돌 개수}}{n(PLSetSetCom)C_2})$$

FoCC는 PLA가 얼마나 컴포넌트 충돌을 발생시키지 않는지를 측정하는 값으로, 아키텍처 드라이버가 아닌 컴포넌트에 초점을 둔다는 점을 제외하고 FoDC와 유사한 방식으로 계산된다. 다음의 식을 이용하여 이미 구한 FoDC와 FoCC의 값을 평균함으로써 FoC을 구한다.

$FoC = FoDC * W_d + FoCC * W_c$, *where* $W_d + W_c = 1$

아키텍처 드라이버 간의 충돌이 PLA 설계에 미치는 영향과 컴포넌트 간의 충돌이 PLA 설계에 미치는 영향은 각각 다르기 때문에, 다른 가중치인 W_d, W_c가 FoDC와 FoCC에 할당된다. 일반적으로 PLA 설계에 아키텍처 드라이버간의 충돌이 미치는 영향이 컴포넌트 간의 충돌이 미치는 영향보다 크기 때문에 두 가중치는 $W_d > W_c$ 로 부여된다.

위의 공식으로 계산된 FoC 값의 범위는 0과 1사이가 된다. FoC 값이 0에 가까울수록 PLA에는 많은 수의 아키텍처 충돌이 발생하여 설계된 PLA가 다소 불안정하고 복잡한 반면에, FoC 값이 1에 가까울수록 PLA에는 적은 수의 아키텍처 충돌이 발생하여 보다 안정적인 PLA를 얻을 수 있게 된다.

4.3 특화 가능성(Tailorability, TAL)

TAL 메트릭은 PLA가 목표 어플리케이션에 맞게 특화될 수 있는지를 측정한다. TAL은 아키텍처 드라이버의 가변성 정도(Degree of Driver-Variability, DDV), 아키텍처 스타일의 가변성 정도(Degree of Style-Variablity, DSV), 컴포넌트의 가변성 정도(Degree of Component-Variability, DCV)의 세 가지 세부 메트릭을 이용하여 측정한다.

$$DDV = \frac{n(OptionalSetD \cup AlternativeSetD)}{n(PLSetD)}$$

DDV는 얼마나 많은 선택 아키텍처 드라이버와 대안 아키텍처 드라이버가 PLA에 설계되어 있는지를 측정하는 값으로, 위의 식과 같이 선택 아키텍처 드라이버와 대안 아키텍처 드라이버의 총 개수를 모든 아키텍처 드라이버의 개수로 나누어 값을 구한다.

DSV와 DCV는 각각 얼마나 많은 가변적인 스타일과 가변적인 컴포넌트가 PLA에 설계되어 있는지를 측정하는 값으로, 아키텍처 드라이버가 아닌 아키텍처 스타일과 컴포넌트에 초점을 둔다는 점을 제외하고 DDV와 유사한 방식으로 계산된다. 다음의 식과 같이 이미 구한 DDV, DSV, DCV의 값에 일정한 가중치를 부여함으로써 TAL을 구한다.

$$DSV = \frac{n(OptionalSetS \cup AlternativeSetS)}{n(PLSetS)}$$

$$DCV = \frac{n(OptionalSetCom \cup AlternativeSetCom \cup OptionalSetComRel \cup AlternativeSetComRel)}{n(PLSetCom)}$$

다음의 식과 같이 이미 구한 DDV, DSV, DCV의 값에 일정한 가중치를 부여함으로써 TAL을 구한다

$TAL = DDV * W_d + DSV * W_s + DCV * W_c$,

where $W_d + W_s + W_c = 1$

아키텍처 드라이버, 아키텍처 스타일, 컴포넌트가 PLA 설계에 미치는 영향은 각각 다르기 때문에, 다른 가중치인 W_d, W_s, W_c 가 DDV, DSV, DCV에 할당된다. 일반적으로 PLA 설계에 아키텍처 드라이버가 미치는 영향이 아키텍처 스타일이 미치는 영향보다 크고, 아키텍처 스타일이 PLA 설계에 미치는 영향이 컴포넌트에 미치는 영향보다 크기 때문에 세 가중치는 $W_d > W_s > W_c$ 순서로 부여된다.

위의 공식으로 계산된 TAL 값의 범위는 0과 1사이가 된다. TAL 값이 0에 가까울수록 PLA의 많은 부분들이 어플리케이션을 유도하는데 특화되기 힘든 반면에, FoC 값이 1에 가까울수록 PLA의 많은 부분들이 어플리케이션을 유도하는데 적용될 수 있어 보다 유연한 PLA를 얻을 수 있게 된다.

4.4 메트릭 결과를 이용한 PLA의 평가

이전 장에서 제시된 메트릭은 요구사항의 준수성, 요구사항 충돌에 대한 안정성, 특화 가능성의 세 가지의 독립적인 측면으로 PLA를 평가하는데 사용된다. 이렇게 측정된 메트릭의 결과에 따라 PLA 설계를 정제할 수 있는데, 예상되는 대표적인 경우는 다음과 같다.

- FoC와 TAL가 높으나 ARC가 저조하다면 PLA 설계를 다시 고려해야 한다. 이는 가변적인 요구사항을 많

이 적용하고 충돌도 적어 한 프로덕트라인에 설계되기 적합하나 공통적인 요구사항이 많이 적용되지 않은 경우이기 때문이다.

- TAL은 프로덕트라인 개발의 초기단계인 스코핑(scoping) 단계에서 정의된 핵심자산이 제공해야 할 가변치와 비교하여 그 정도가 평가되어야 한다. TAL는 특화 가능성의 정도를 표현할 뿐이지 적용하려는 PL에 대한 유효성을 검증하는 것은 아니기 때문이다.
- TAL의 값이 너무 높아서 PL에서 묵시적으로 인식되는 가변치 허용 정도를 초과한다면, 어떤 가변치는 포기되어야 하고 이로 인해 ARC는 낮아지게 된다. 따라서, ARC와 TAL간의 교환(trade-off)이 필요하다.
- FoC가 높으나 분석된 요구사항 충돌 지점이 서로 다른 멤버간에 나타난 경우는 충돌로 인해 안정성이 낮아지지 않는다.

5. 사례 연구

이 장에서는 사례 연구를 수행함으로써, 앞에서 정의한 평가 메트릭이 어떻게 적용되는지를 보여준다. 적용되는 PL은 도서관 대여 시스템과 자동차 대여 시스템을 멤버로 포함하는 대여 관리 도메인이다. 두 멤버들로부터 수집한 요구사항을 먼저 공통성과 가변성으로 분류한 후, 이 정보를 이용하여 공통적 또는 가변적인 아키텍처 드라이버를 식별한다. 이 결과, 유지보수성은 필수 드라이버, 신뢰성, 접근성, 성능과 데이타 무결성은 선택 드라이버로, 방화벽 또는 로깅을 이용한 보안성은 대안 드라이버로 식별되었다. 그림 5는 모듈 뷰 관점에서 대여 관리 도메인을 위한 PLA를 나타낸 것이다.

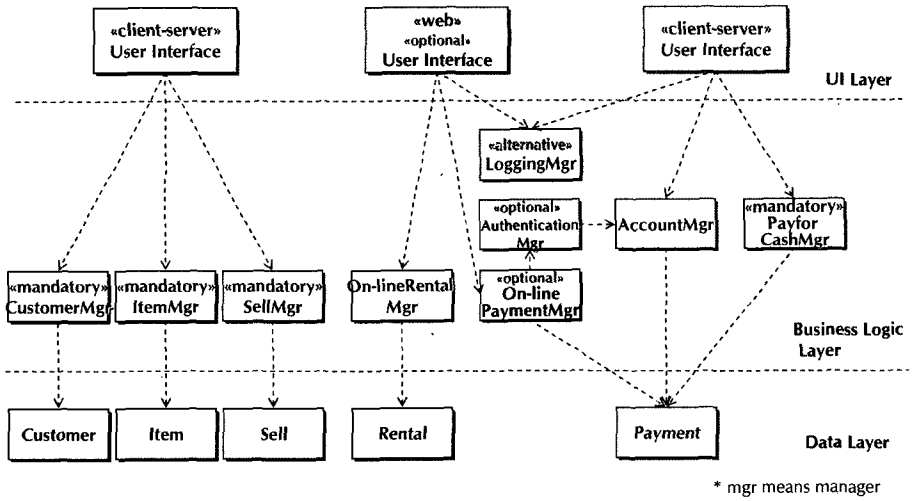

그림 5 대여 관리 도메인을 위한 PLA

표 1 대여 관리 도메인을 위한 아키텍처 요소와 관계에 대한 명세서

아키텍처 드라이버			스타일 / 패턴	컴포넌트	
				드라이버에서유도된 컴포넌트	기능적 요구사항에서 유도된 컴포넌트
필수	유지보수성(Maintainability)		계층스타일	없음	• Customer • Item • Sell • Rental • Payment • CustomerMgr • ItemMgr • SellMgr • On-lineRentalMgr • AccountMgr • PayforCashMgr • User Interface for C/S
선택	신뢰성(Reliability)		파이프와 필터 스타일	• uthenticationMgr • n-linePaymentMgr	
	접근성(Accessibility)		없음	• ser Interface for Web	
	성능(Performance)		분산 데이타 스타일	없음	
	데이타 무결성(Data Integrity)		공유 데이타 스타일	없음	
대안	보안성(Security)	방화벽 이용	없음	• 방화벽	
		로깅 이용	없음	• LoggingMgr	

위의 PLA 설계는 모듈 뷰 관점에서 표현된 것이기 때문에, 계층 스타일로만 표현되어 있고 총 열 여섯 개의 컴포넌트가 계층 스타일에 할당되어 PLA를 구성한다. 표 1은 모듈 뷰 뿐만 아니라 그 외의 뷰에서 표현된 PLA 요소와 이들의 유도 관계에 대해 상세하게 명세한 것이다. 예를 들어, 선택 드라이버인 '신뢰성'은 PLA에서 '파이프와 필터 스타일'로 해결되며, 이 스타일을 선택함으로써 'AuthenticationMgr' 컴포넌트와 'On-line-PaymentMgr' 컴포넌트가 유도된다.

이제 위의 표 1을 이용하여 ARC, FoC, TAL을 계산함으로써 PLA를 평가한다. 표 2는 각 메트릭과 메트릭을 적용한 결과를 보여준다. ARC를 유도하기 위해 세 개의 세부 메트릭 DMDC, DODC, DADC을 계산하였고, FoC를 유도하기 위해 두 개의 서브 메트릭 FoDC

표 2 평가 메트릭과 적용 결과

메트릭	적용 및 결과
DMDC	$\frac{(n(SetD_{MandatorySetS} \cup SetD_{MandatorySetCom}))}{n(MandatorySetD)} = \frac{1}{1} = 1$ * $SetD_{MandatorylSetS}$ = {유지보수성} * $SetD_{MandatorySetCom}$ = { } * MandatorySetD = {유지보수성}
DODC	$\frac{(n(SetD_{OptionalSetS} \cup SetD_{OptionalSetCom}))}{n(OptionalSetD)} = \frac{4}{4} = 1$ * $SetD_{optionalSetS}$ = {신뢰성, 성능, 데이타 무결성} * $SetD_{optionalSetCom}$ = {신뢰성, 접근성} * OptionalSetD = {신뢰성, 접근성, 성능, 데이타 무결성}
DADC	$DADC = \frac{(n(SetD_{AlternativeSetS} \cup SetD_{AlternativeSetCom}))}{n(AlternativeSetD)} = \frac{2}{2} = 1$ * $SetD_{AlternativeSetS}$ = { } * $SetD_{AlternativeSetCom}$ = {방화벽을 이용한 보안성, 로깅을 이용한 보안성} * AlternativeSetD = { 방화벽을 이용한 보안성, 로깅을 이용한 보안성}
ARC	(DMDC + DODC + DADC) / 3 = 3/3 = 1
FoDC	$FoDC = 1 - (\frac{\text{드라이버 충돌 개수}}{n(PLSetD)C_2}) = 1 - \frac{1}{15} = 0.93$ * 자동차 대여 시스템의 여러 사용자는 제한된 시간 내에 데이타를 접근하기 원하기 때문에(성능), 데이타가 한 곳이 아니라 여러 곳에 분산되어 저장되어야 한다. 반면에, 도서관 대여 시스템에서는 모든 데이타가 일관성 있게 유지되는 것을 원하기 때문에(데이타 무결성), 데이타가 한 곳에 저장되어야 한다. 이 두 멤버가 포함된 대여 관리 도메인을 위한 PLA는 충돌을 일으키는 두 가지 드라이버를 모두 설계해야 한다.
FoCC	$FoCC = 1 - (\frac{\text{컴포넌트 충돌 개수}}{n(PLSetSetCom)C_2}) = 1$ * 충돌을 일으키는 컴포넌트 없음.
FoC	FoC = (FoDC+ FoCC) / 2 = (1 + 0.94) / 2 = 0.965
DDV	$DDV = \frac{(n(OptionalSetD \cup AlternativeSetD))}{n(PLSetD)} = \frac{5}{6} = 0.83$ * OptionalSetD = {신뢰성, 접근성, 성능, 데이타 무결성} * AlternativeSetD(super driver) = {보안성} * PLSetD = {유지보수성, 신뢰성, 접근성, 성능, 데이타 무결성, 보안성}
DSV	$DSV = \frac{(n(OptionalSetS \cup AlternativeSetS))}{n(PLSetS)} = \frac{3}{4} = 0.75$ * OptionalSetS = {파이프와 필터 스타일, 분산 데이타 스타일, 공유 데이타 스타일} * AlternativeSetS(super driver) = {} * PLSetS = {계층 스타일, 파이프와 필터 스타일, 분산 데이타 스타일, 공유 데이타 스타일}
DCV	$DCV = \frac{(n(OptionalSetCom \cup AlternativeSetCom))}{n(PLSetCom)} = \frac{5}{17} = 0.29$ * OptionalSetCom = {User Interface for Web, AuthenticationMgr, On-linePaymentMgr} * AlternativeSetCom(super component) = {LoggingMgr, 방화벽} * PLSetCom = {User Interface for C/S, User Interface for Web, CustomerMgr, ItemMgr, SellMgr, On-lineRentalMgr, LoggingMgr, AuthenticationMgr, On-linePaymentMgr, AccountMgr, PayforCashMgr, Customer, Item, Sell, Rental, Payment, 방화벽}
TAL	TAL = DDV * W_d + DSV * W_s + DCV * W_c = 0.83 * 0.5 + 0.75 * 0.3 + 0.29 * 0.2 = 0.415 + 0.225 + 0.058 = 0.698 * 드라이버, 스타일, 컴포넌트가 아키텍처 설계에 미치는 영향을 고려하여 W_d , W_s , W_c에 각각 0.5, 0.3, 0.2을 부여하였다.

와 FoCC를 계산하였으며, TAL을 유도하기 위해 세 개의 서브 메트릭인 DDV, DSV, DCV를 계산하였다.

계산된 결과에서는 ARC는 1로 도출되어 PLA에 적용될 수 있는 모든 요구 사항이 반영된 것으로 나타났다. 그러나 FoC로 약간의 요구사항간의 충돌이 있는 것으로 보여졌다. TAL의 경우 프로덕트라인 개발에서 허용되는 가변치의 값을 초과하게 된다면, 일부 가변치는 포기되어야 하고 이로 인해 ARC가 낮아질 수 있다.

6. 결 론

PLA는 재사용 공학으로 새롭게 대두되고 있는 PLE의 핵심 자산 중에 중요한 요소 중 하나이다. PLA는 PL의 여러 어플리케이션들이 공통으로 사용하는 범용 아키텍처이기 때문에, PLA를 평가하는 방법은 한 시스템을 위한 아키텍처를 평가하는 방법과 다르게 정의되어야 한다. 만약 PLA와 한 시스템을 위한 아키텍처의 차이점이 구체적으로 다루어지지 않는다면, PLA를 평가하는 것은 결국 PLE에서 어려운 작업이 된다.

본 논문에서는, 먼저 PLA의 평가대상이 되는 요소인 아키텍처 드라이버, 아키텍처 스타일, 컴포넌트와 컴포넌트 간 관계에 대해 기술하였고, PLA 설계 시에 중요하게 다루어져야 하는 두 가지 이슈인 '가변성 파급에 대한 이슈'와 '아키텍처 요소 간의 충돌에 대한 이슈'에 대해 상세히 알아보았다. 그리고, 식별한 두 가지 이슈를 중점으로 PLA를 정량적으로 평가하기 위한 세 가지 메트릭을 정의하였다. 본 논문에서 정의된 메트릭은 아키텍처 요구사항에 대한 준수성(ARC), 아키텍처 충돌에 대한 안정성(FoC), 특화 가능성(TAL)이며, 이 세 메트릭의 결과를 이용하여 PLA를 어떻게 평가할 수 있는지를 알아보았다. 마지막으로, 본 논문에서 정의한 메트릭이 어떻게 적용되는지를 알아보기 위해 간단한 사례 연구를 수행하였다.

식별된 두 가지 이슈와 이슈로부터 도출한 PLA 평가 메트릭은 설계된 PLA가 원래 요구사항을 제대로 만족하였는지, 요구사항을 만족시키기 위해 효과적으로 설계되었는지를 평가하기 위한 것이다. 그러므로, 평가 결과를 기준으로 PLA 설계를 다시 수행해야 하는지 또는 PLA 설계 다음 단계로 진행해도 되는지의 여부를 결정할 수 있어 높은 품질의 PLA를 좀더 효율적으로 설계하는데 도움이 된다. 그리고, 시나리오와 아키텍처 설계자의 경험을 기반으로 하는 평가 방법이 아닌 메트릭을 기반으로 PLA를 평가하기 때문에, 정량적이며 정확하고 객관적인 평가 결과를 얻을 수 있으며, PLA 평가를 좀더 체계적으로 적용할 수 있게 될 수 있을 것이라고 예상된다.

참 고 문 헌

[1] Bosch, J., *Design and Use of Software Architectures*. Addison-Wesley, 2000.

[2] Anastasopoulos, M., Bayer, J., Flege, O., and Gacek, C., "A Process for Product Line Architecture Creation and Evaluation PuLSE-DSSA-version 2.0", *Technical Report, No. 038.00/E*, IESE, June 2000.

[3] Clements, P., et al., *Evaluating Software Architecture*, Addison Wesley, 2002.

[4] Etxeberria, L. and Sagardui, G, "Product-Line Architecture: New Issues for Evaluation," *Proceedings of SPLC 2005, LNCS 3714*, Springer-Verlag Berlin Heidelberg, 2005.

[5] Olumofin, F. and Misic, V., "Extending the ATAM Architecture Evaluation to Product Line Architecture," *Proceedings of Fifth Working IEEE/IFIP Conference on Software Architecture (WICSA 5)*, To appear.

[6] Niemelia, E., Matinlassi, M., Taulavuori, A., "Practical Evaluation of Software Product Family Architecture," *Proceedings of SPLC, LNCS 3154*, Springer-Verlag Berlin Heidelberg, 2004.

[7] Bayer, J. et al., "PuLSE: A Methodology to Develop Software Product Lines," *Proceeding of Symposium on Software Reusability '99*, May 1999.

[8] Obbink, H. et al., "COPA: A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electric Products," *Tutorial for the First Software Product Line Conference (SPLC1)*, Aug. 2000.

[9] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," VTT Technical Research Center of Finland, *Proceedings of ESPOO2002*, 2002.

[10] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.

[11] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Standard P1471)* IEEE Architecture Working Group (AWG); 2000.

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

장 수 호
정보과학회논문지 : 소프트웨어 및 응용
제 33 권 제 2 호 참조

라 현 정
2003년 경희대학교 우주과학과 이학사
2006년 숭실대학교 컴퓨터학과 공학석사
2006년~현재 LG-CNS. 관심분야는 제품계열 공학(PLE), 소프트웨어 아키텍처

김 수 동
정보과학회논문지 : 소프트웨어 및 응용
제 33 권 제 1 호 참조