제품계열공학 핵심자산의 범용 아키텍처 구성요소

라현정, 장수호, 김수동
숭실대학교 대학원 컴퓨터학과
{hjla, shchang}@otlab.ssu.ac.kr, sdkim@ssu.ac.kr

Key Elements of Generic Architecture in PLE Core Assets

Hyun Jung La, Soo Ho Chang, and Soo Dong Kim
Dept. of Computer Science, Soongsil University

요 약

제품 계열 공학(Product Line Engineering, PLE)는 패밀리 멤버들의 공통성과 가변성을 분석하여 만든 핵심 자산을 특화시켜 어플리케이션을 개발함으로써 재사용성과 이용가능성을 증대시키는 접근 방법이다 [1]. 핵심 자산은 제품 계열에 속하는 패밀리 멤버들이 어플리케이션을 만드는데 기초가 되는 모든 자산을 포함하며, 아키텍처, 컴포넌트 등이 포함될 수 있다. 범용 아키텍처는 패밀리 멤버들이 공통적으로 사용할 수 있는 아키텍처로, 제품 계열에 속하는 제품들의 구조를 정의하고 컴포넌트의 인터페이스 명세를 제공하여 컴포넌트만큼 중요한 재사용 단위이다 [2]. 본 논문에서는 대표적인 PLE 방법론에서 정의한 제품 계열 아키텍처와 일반 소프트웨어 아키텍처를 비교하여 범용 아키텍처에 포함되는 요소들을 선정하고, 메타 모델을 이용하여 범용 아키텍처 구성요소와 구성요소간 관계를 명확히 정의함으로써, 개념적인 아키텍처를 보다 실용적으로 설계하는데 도움이 되게 하고자 한다.

1. 서론

제품 계열 공학(Product Line Engineering, PLE)는 패밀리 멤버들의 공통성과 가변성을 분석하여 만든 핵심 자산을 특화시켜 어플리케이션을 개발함으로써 재사용성과 이용가능성을 증대시키는 접근 방법이다. 핵심 자산은 제품 계열에 속하는 패밀리 멤버들이 어플리케이션을 만드는데 기초가 되는 모든 자산을 포함하며, 범용 아키텍처, 컴포넌트, 결정 모델이 포함된다. 범용 아키텍처는 패밀리 멤버들이 공통적으로 사용할 수 있는 아키텍처로, 제품 계열에 속하는 제품들의 구조를 정의하고 컴포넌트의 인터페이스 명세를 제공하여 컴포넌트만큼 중요한 재사용 단위이다. 그러므로 범용 아키텍처를 중심으로 핵심 자산을 설계하면 더욱 효과적이다 [3]. 기존의 연구에서도 범용 아키텍처의 중요성은 언급하지만, 범용 아키텍처에 대한 정의와 구성요소 명세가 명확하지 않는다 [4]. 본 논문에서는 범용 아키텍처 구성요소와 구성요소간 관계를 명확히 정의함으로써, 개념적인 아키텍처를 보다 실용적으로 설계하는데 도움이 되게 하고자 한다.

본 논문에서는 2장과 3장에서 기존 연구에서 제시된 범용 아키텍처와 일반 아키텍처를 조사하고, 이를 기반으로 4장에서 범용 아키텍처의 메타 모델을 나타내고, 범용 아키텍처의 구성 요소에 대해 상세히 설명한다.

2. 관련 연구

PuLSE-DSSA는 PuLSE 방법론에서 제시한 제품 계열 아키텍처 설계를 위한 접근 방법으로 기능적, 비기능적 요구사항을 나타내는 시나리오(Scenario) 중 중요한 시나리오를 선정하여 점진적으로 참조 아키텍처를 생성한다 [3][5]. PuLSE-DSSA 아키텍처의 구성요소에 대해 유추는 가능하지만 명확하게 언급하지 않는다.

Bosch의 제품 계열 공학 아키텍처는 아키텍처에 포함되는 컴포넌트들과 관계로 이루어져 있다 [6]. 범용 아키텍처는 제품 계열에 속하는 모든 제품과 회처를 포함하여 제품마다 다른 점을 '가변성'을 이용하여 나타내는 관점과 제품 계열의 모든 패밀리들이 공통으로 사용하는 제품과 회처만 포함하는 관점으로 표현된다. Bosch는 비기능적인 요소도 고려하였으나, 아키텍처에 포함되는 구성요소와 구성요소가 관계에 대한 정의가 미비하다.

COPA에서 명시한 아키텍처는 컴포넌트와 컴포넌트간 관계를 포함하는 시스템 기본 단위로, 개발시 도메인 내에 선택되지 않은 패밀리도 고려한 참조 아키텍처(Reference Architecture)와 도메인 내 선택된 패밀리를 위한 패밀리 아키텍처(Family Architecture)로 분류한다 [7]. QADA는 기능적 비기능적인 요구사항을 아키텍처로 변환하는 체계적인 방법을 제공하여 아키텍처를 설계하고 분석하는 방법을 제시한다 [8]. 아키텍처는 구조적인 뷰(Structural View), 행위적인 뷰(Behavioral View), 배치 뷰(Deployment View를 이용하여 설계, 분석된다. COPA와 QADA에서는 소프트웨어 아키텍처에서 강조하는 뷰도 고려하지만, 전체적으로 프로세스 위주로 아키텍처 정의와 구성 요소뿐만 아니라 요소간 관계에 대해 명확하게 언급하지 않는다.

KobrA[9] 방법론은 구체적으로 아키텍처를 언급하고 있지 않지만, 컴포넌트간 포함관계를 나타내는 컨테인먼트 트리(Containment Tree)와 컨텍스트 실현(Context Realization)으로 아키텍처를 유추할 수 있다.

3. 대표적 PLE 기법들의 범용 아키텍처 비교

본 장에서는 2장에서 간략하게 설명한 여러 PLE 방법론에서 언급한 제품 계열 아키텍처의 구성요소를 비교하여 본 논문에서 제시할 범용 메타모델의 구성 요소를 도출하는데 기본정보로 활용한다. 표 1에서는 메타모델의 구성요소를 도출하기 위해 2장에서 언급한 제품 계열 아키텍처 구성요소를 간략히 나열한다.

표 1 제품 계열 아키텍처 구성요소 비교

	Elements
PuLSE [3][10]	아키텍처 프로토타입, 명세(모델), 가변성 모델
Bosch [6]	참조 컨텍스트, 구조 (컴포넌트, 관계)
COPA [7]	컴포넌트, 관계, 뷰
QADA [8]	컴포넌트, 관계, 뷰
KobrA [9]	컴포넌트, 관계

표 1에서 비교한 바와 같이, 본 논문에서 비교한 5개의 PLE방법론의 제품 계열 아키텍처의 구성요소로 공통적으로 컴포넌트와 컴포넌트 간 관계를 명시하고 있다.

PuLSE 방법론에서는 구체적으로 컴포넌트와 관계를 명시하고 있지 않지만 명세(모델)에 그 내용이 내포되어 있고, 다른 방법론과 달리 가변성 모델을 아키텍처의 구성요소로 보고 있다. Bosch의 제품 계열 아키텍처는 시스템과 시스템의 환경을 인터페이스로 정의한 참조 컨텍스트 외에 아키텍처를 분해하는 과정에서 생성된 컴포넌트와 컴포넌트간 관계를 표현하는 구조(Structure)가 아키텍처의 구성요소이다. COPA와 QADA는 아키텍처의 구성요소를 컴포넌트와 컴포넌트간 관계로만 명시하는 것 외에 뷰도 고려한다. KobrA 방법론에서는 구체적으로 아키텍처라고 명시되어 있지 않지만, 컨테인먼트 트리와 컨텍스트 실현으로 유추할 수 있고, 구성요소는 컴포넌트와 컴포넌트 간 관계로 이루어져 있다.

그러나, 일부 PLE 방법론의 제품 계열 아키텍처는 SEI에서 제시한 아키텍처나 IEEE, P1471 아키텍처에서 중요시하는 뷰타입과 스타일을 구성요소에 포함시키지 않는다 [4][11]. 사용자의 기능적인 요구사항 외 비기능적인 요구사항(품질 요소)을 효과적으로 설계하기 위해 소프트웨어 아키텍처는 뷰와 스타일을 사용한다. 아키텍처를 사용자의 요구에 맞는 특정한 시각으로 묘사하기 위해 뷰타입을 선택하고, 각 뷰타입에 존재하는 여러 스타일 중 하나를 선택하여 소프트웨어 아키텍처가 설계된다. SEI에서는 뷰타입을 Module 뷰타입, Component-and-Connector(C&C) 뷰타입, Allocation 뷰타입으로 분류한다.

그림 1은 제품 계열 아키텍처의 구성요소와 소프트웨어 아키텍처의 구성요소를 비교한 결과 본 논문에서 제시할 메타 모델의 요소를 도출한 결과로서, 제품 계열 아키텍처에서 유도된 컴포넌트, 컴포넌트간 관계, 결정 모델과 일반 소프트웨어 아키텍처에서 유도된 뷰, 스타일로 구성된다.

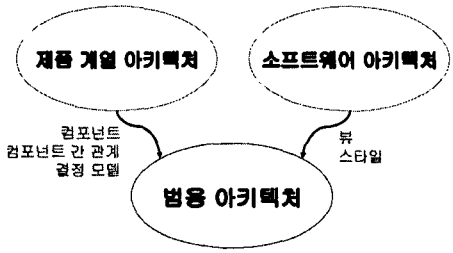

그림 1. 범용 아키텍처 구성요소

4. 범용 아키텍처 참조 모델

본 장에서는 3장에서 비교한 내용을 기반으로 범용 아키텍처의 메타 모델을 4.1에서 제시하고, 4.2에서는 본 논문에서 제시한 메타 모델의 구성요소에 대해 설명한다.

4.1. 메타 모델

본 절에는 표 1에서 살펴본 제품 계열 공학 방법론의 범용 아키텍처의 구성 요소와 일반 아키텍처의 구성 요소를 기반으로 그림 2과 같은 메타 모델을 제안한다.

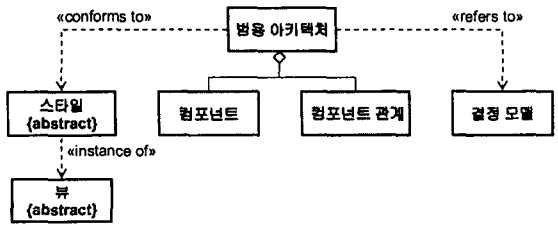

그림 2. 범용 아키텍처 메타 모델

그림 2과 같이 범용 아키텍처는 컴포넌트와 컴포넌트간 관계의 두 개 요소로 이루어져 있다. 컴포넌트와 컴포넌트 간 관계는 여러 패밀리 멤버에 공통적인 부분 외에 가변적인 부분도 포함하고 있으므로 가변성 정보를 갖고 있는 아키텍처 결정 모델을 참조한다. 따라서, 컴포넌트와 컴포넌트간 관계를 포함하는 범용 아키텍처는 결정 모델과 '*refers to*' 관계를 갖는다. 범용 아키텍처는 추상적 요소인 스타일에 따라 여러 범용 아키텍처가 유도되고, 여러 스타일의 조합에 의해 한 범용 아키텍처가 유도되므로 범용 아키텍처와 스타일 사이에는 '*conforms to*' 관계를 갖는다. 아키텍처 설계할 때 사용자가 원하는 요구사항에 맞게 선택되는 뷰타입는 여러 스타일을 포함한다. SEI에서 제시한 뷰타입 중 Allcocation 뷰타입은 어플리케이션과 해당 어플리케이션의 하드웨어 등 실행 환경간의 관계를 나타낸 것으로, 여러 패밀리 멤버의 공통성이 상대적으로 적으므로 본 논문의 메타모델에 표현된 뷰타입은 Allocation 뷰타입을 제외하고 Module 뷰타입과 C&C 뷰타입로 분류될 수 있다.

4.2. 구성 요소별 정의

본 절에는 4.1에서 제시한 메타 모델의 구성 요소에 대해 자세히 설명한다. 먼저 범용 아키텍처의 물리적인 구성요

소에 대해 언급하고, 그 후에 물리적인 구성요소는 아니지만 범용 아키텍처와 관계를 맺고 있는 요소에 대해 설명한다.

범용 아키텍처는 물리적인 요소인 컴포넌트와 컴포넌트간 관계를 포함하고 범용 아키텍처와 이들 간의 관계는 그림 2와 같이 포함(Aggregation)관계를 갖는다.

컴포넌트(Component)는 기능을 구현하고 있는 단위로 아키텍처를 여러 작은 단위로 분해하는 과정에서 추출된다. 컴포넌트는 아키텍처의 기능적인 요구사항과 비기능적인 요구사항을 모두 구현한 단위이다. 컴포넌트는 데이터와 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.

관계(Relationship)은 컴포넌트가 고유한 기능을 완벽하게 수행하기 위해 필요한 요소로, 하나의 컴포넌트는 스스로 고유한 기능을 수행하지 못하는 경우에 다른 컴포넌트와 관계를 맺는다. 컴포넌트간 관계로는 한 컴포넌트가 다른 컴포넌트의 기능을 사용하거나 실행시 다른 컴포넌트의 메시지를 호출하는 관계인 의존 관계(Dependency)와 한 컴포넌트가 그보다 작은 단위의 컴포넌트를 포함하는 포함 관계(Composition)관계가 있다.

그림 2와 같이 직접적으로 범용 아키텍처에 포함된 물리적인 요소 외에 범용 아키텍처는 개념적인 요소인 결정 모델을 참조하고, 추상적인 요소인 스타일과는 일치하는 관계를 갖는다. 스타일은 또 다른 추상적인 요소인 뷰의 인스턴스이므로 스타일과 뷰 사이에 관계가 존재한다.

결정 모델(Decision Model)에는 제품 계열에 속한 여러 패밀리 멤버에 의해 재사용되기 위해 가변적인 부분을 포함하고, 이는 패밀리 멤버의 가변적인 부분이 범용 아키텍처에 실현화(Realize)되어야 한다. 이렇듯, 가변성이 범용 아키텍처의 기본요소에 내포되어 있어 범용 아키텍처가 결정 모델을 참조(refers to)하고, 범용 아키텍처에 의해 참조된 결정 모델은 범용 아키텍처의 물리적인 구성요소가 아니라 개념적인 요소가 된다. 결정 모델에 포함되는 가변성은 컴포넌트와 컴포넌트간 관계, 아키텍처에서 생길 수 있다. 소프트웨어 아키텍처가 범용 아키텍처와 크게 다른 점은 바로 범용 아키텍처가 결정 모델을 포함한다는 것이다.

스타일(Style)은 아키텍처가 사용자의 기능적인 요구사항 외에 비기능적인 요구사항인 품질 요소에 맞게 설계되기 위해 사용되는 요소이다. 소프트웨어 아키텍처는 가장 먼저 소프트웨어를 바라보는 적절한 뷰를 선택하고, 그 뷰에 속하는 스타일을 선택한 후, 스타일에 맞게 컴포넌트와 컴포넌트 간 관계를 추출하는 과정을 거쳐 설계된다 [11]. 한 뷰에 속하는 여러 스타일에 따라서 범용 아키텍처에 속하는 컴포넌트와 관계가 결정되므로 여러 스타일에 일치(conforms to)하는 범용 아키텍처가 설계되고, 그 과정에서 추출된 컴포넌트와 컴포넌트 관계에는 스타일이 반영된 것이다. 그러므로, 뷰타입과 스타일은 범용 아키텍처를 이루는 물리적인 구성요소가 아니라 추상적인 요소이다.

5. 결론 및 향후 연구과제

범용 아키텍처는 재사용할 수 있는 부분을 정의하여, 공통적인 컴포넌트 외에 특정 패밀리 멤버를 위한 컴포넌트도 포함하고 있으므로 범용 아키텍처를 중심으로 핵심 자산을 설계하면 더욱 효과적이다. 지금까지 여러 기존 연구에서 아키텍처의 구성요소는 언급이 되었으나, 본 논문에서는 대표적인 PLE 방법론에서 정의한 제품 계열 아키텍처와 일반 소프트웨어 아키텍처를 비교하여 본 논문에서 제시할 메타 모델에 포함되는 요소들을 선정하였다. 그리고 선정된 요소들을 기반으로 범용 아키텍처의 메타 모델을 제시하여 메타 모델에 포함된 요소간의 관계를 보다 명확히 정의하였다. 기존 연구에서 개념적으로 정의가 된 범용 아키텍처의 요소와 요소간의 관계를 명확히 정의함으로써 보다 실용적으로 아키텍처를 설계하고, 나아가 핵심 자산도 효과적으로 설계하는데 도움이 될 것으로 기대가 된다.

앞으로 본 논문에서 제시한 메타모델에 PLE에서 중요한 개념인 가변성 모델과 아키텍처에서 중요하게 여기는 품질 요소에 대한 개념을 포함해야 할 것이며, 더 나아가 이 논문에서 제시한 메타 모델을 이용하여 더욱더 실용적인 방법으로 아키텍처를 설계하기 위하여 아키텍처설계 프로세스와 지침을 대한 연구가 필요하다.

6. 참고문헌

[1] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns*, Addison Wesley, Aug. 2001.

[2] Bass, L., Clements, P., Kazman, R.. *Software Architecture in Practice*, Addison Wesley, 1998.

[3] Bayer, J., flege, O., and Gacek, C., "Creating Product Line Architectures," *proceeding of IW-SAPF-3, LNCS 1951*, Springer-Verlag Berlin Heidelberg, 2000.

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

[5] 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.

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

[7] 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.

[8] Matinlassi, M., Niemelä, E., and Dobrica. L., *Quality-driven architecture design and quality analysis method*, VTT Publications, pp. 128, 2002.

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

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

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