DOI: 10.3745/KIPSTD.2008.15-D.6.803

다중 관점 제품계열아키텍처의 가변성 관리 및 일관성 검사를 위한 특성 지향 접근방법

이 관 우†

요 약

제품계열아키텍처는 제품에 따라 선택될 수 있는 가변요소를 포함하고 있는 아키텍처이다. 제품계열아키텍처부터 특정 제품을 위한 유효한 아키텍처를 유도하기 위해서는 제품계열아키텍처 내의 가변요소들을 체계적으로 관리해야 한다. 본 논문에서는 특성모델과 제품계열아키텍처 모델간의 명시적인 대응관계를 통해서 제품계열아키텍처의 가변성을 관리한다. 하지만, 이들 모델 간의 대응관계가 올바르지 않거나, 제품계열아키텍처의 구성요소들 간에 일관성이 없다면, 제품계열아키텍처의 가변성 관리가 올바르게 이루어지지 않게 된다. 따라서 본 논문에서는 먼저, 제품계열아키텍처를 개념, 프로세스, 배치, 모듈의 네 가지 관점의 모델로 정의하고, 특성 모델과 이들 모델 사이의 대응관계를 정형적으로 정의한다. 이를 바탕으로 제품계열아키텍처의 올바른 가변성 관리를 위해서, 제품계열아키텍처 모델의 일관성, 다른 관점의 아키텍처 모델간의 일관성, 특성모델과 제품계열아키텍처 모델간의 일관성 검사를 위한 규칙을 정의한다. 이러한 일관성 규칙은 제품계열아키텍처로부터 유효한 제품 아키텍처를 유도하기 위한 이론적 기반을 제공한다.

키워드 : 소프트웨어 제품계열 공학, 가변성 모델, 특성 모델, 일관성 분석, 아키텍처 뷰

A Feature-Oriented Approach to Variability Management and Consistency Analysis of Multi-Viewpoint Product Line Architectures

Kwanwoo Lee†

ABSTRACT

Product line architectures include variable parts to be selected according to product specific requirements. In order to derive architectures that are valid for a particular product from product line architectures, variabilities of product line architectures must be systematically managed. In this paper, we adopt an approach to variability management of product line architectures through an explicit mapping between a feature model and product line architecture models. If this mapping is incorrect or there exists inconsistency among product line architectural elements, variabilities of product line architectures cannot be managed correctly. Therefore, this paper formally defines product line architectural models in terms of conceptual, process, deployment, and module views, and mapping relationships between the feature model and the architectural models. Consistency rules for correct variability management of product line architectures are defined in terms of consistency in each of product line architecture model, consistency between different architectural view models, and consistency between a feature model and product line architectural models. These consistency rules provide a theoretical foundation for deriving valid product architecture from product line architectures

Keywords : Product Line Engineering, Variability Model, Feature Model, Consistency Analysis, Architectural Views

1. 서 론

제품계열공학(product line engineering) [1]은 최근에 각광받고 있는 소프트웨어 재사용 방법으로, 유사한 제품 개발에 재사용될 수 있는 제품계열자산(예, 아키텍처 및 컴포넌트)을 미리 개발한 후에, 이를 체계적이고 계획적으로 재사용함으로써 다양한 고객의 요구에 맞는 제품을 저비용으로 적기에 생산하는 것을 목표로 한다.

"제품계열아키텍처(product line architecture)"는 여러 제품계열자산 중에 핵심적인 요소로서, 제품계열 범위 안에 있는 모든 제품 개발에 공통으로 재사용될 수 있는 부분뿐만 아니라, 제품에 따라 선택될 수 있는 가변요소도 포함하고 있다. 따라서 제품계열아키텍처로부터 특정 제품을 위한 *유효한* 제품아키텍처(product architecture)를 개발하기 위해

※이 논문은 2005년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2005-003-D00329).
† 정 회 원 : 한성대학교 정보시스템공학과 조교수
논문접수 : 2008년 9월 8일
수 정 일 : 1차 2008년 10월 9일
심사완료 : 2008년 10월 11일

서는 제품계열아키텍처의 가변요소들을 체계적으로 관리해야 한다.

한편, 일반적으로 시스템의 복잡도를 해결하기 위해서는 관심분리(separation of concerns) 원칙에 입각하여 다중 관점으로 아키텍처를 분리하여 설계하는 것이 필요한데, 단일 제품을 위한 아키텍처 설계에 비해서 복잡도가 한층 더 높은 제품계열아키텍처의 설계에서는 다중 관점의 아키텍처 설계가 필수적인 요소이다. 다중 관점으로 분리하여 제품계열아키텍처를 설계하는 경우에 특히 각 관점의 아키텍처가 가지는 가변요소들을 효율적이고 체계적으로 관리할 수 있어야 한다.

제품계열아키텍처의 가변요소를 관리하는 방법은 크게 두 가지가 있다. 첫 번째 방법은 제품계열아키텍처 모델 안에 가변요소 관리를 위한 모델링 수단을 포함시키는 것이다. 가령, 두 개의 컴포넌트들 중에서 하나만 특정 제품을 위해 선택될 수 있는 경우에는 두 컴포넌트를 가변요소로 정의하고 이들이 한 제품을 위해서 오로지 하나만 선택될 수 있도록 택일 가변요소(alternative variant)로 모델링 된다. 두 번째 방법은 제품계열 내의 가변성을 특성(feature) 관점에서 분석하고 모델링 한 특성모델 (feature model)을 이용하여 제품계열아키텍처의 가변요소를 관리하는 방법이다. 즉, 제품계열 아키텍처 모델은 가변요소에 대한 구분만 할 뿐, 이들에 대한 관리는 제품계열아키텍처 모델과 대응된 특성모델을 이용하여 진행된다. 예를 들면, 이전 예의 두 컴포넌트는 가변요소로서 제품계열아키텍처 모델에 표시되고, 각 가변요소는 특성모델 내의 택일 특성(alternative feature)과 대응시킨다. 따라서 택일 특성 중에 하나가 선택됨에 따라 대응되는 가변 컴포넌트가 선택되는 방식을 취한다. 첫 번째 방법은 다중 관점의 아키텍처 모델 안에 유사한 가변성 관리 방식이 중복적으로 포함되는 문제가 있는 반면에, 두 번째 방법은 특성모델을 통해서 다중 관점의 아키텍처 모델들의 가변성이 동일한 방식으로 관리된다는 장점이 있다. 이를 위해서는 아키텍처 모델의 각 가변요소와 특성 모델의 대응관계가 명시적으로 나타내야 한다.

본 논문에서는 특성모델과 제품계열아키텍처 모델과의 명시적인 대응관계를 통해서 제품계열아키텍처의 가변성을 관리하는 방식을 채택한다. 이는 다양한 관점의 아키텍처 가변성뿐만 아니라 상세 설계 및 구현 단계에서 나타나는 가변성까지 하나의 가변성 관리 모델인 특성모델을 이용하여 효율적으로 관리할 수 있기 때문이다.

하지만, 특성모델과 제품계열아키텍처 모델과의 대응관계가 올바르게 이루어지지 않는다면 특성모델로부터 관리되는 제품계열아키텍처의 가변성도 올바르게 이루어지지 않게 된다. 따라서, 본 논문에서는 먼저, 제품계열아키텍처를 개념, 프로세스, 배치, 모듈로 구분되는 네 가지 관점의 모델로 정의하고, 특성모델과 이들 모델 사이의 대응관계를 정형적으로 정의한다. 이를 바탕으로 제품계열아키텍처의 올바른 가변성 관리를 위해서, 제품계열아키텍처 모델의 일관성, 다른 관점의 아키텍처 모델간의 일관성, 특성모델과 제품계열아키텍처 모델간의 일관성 검사를 위한 규칙을 정의한다.

본 논문의 구성은 다음과 같다. 2장에서는 기반 연구로서 기존의 특성모델과 제품계열아키텍처의 개념에 대해 간단히 요약하고 3장에서는 특성모델과 대응 관계를 가진 다중 관점의 제품계열아키텍처 모델을 정형적으로 정의한다. 4장에서는 제품계열아키텍처 모델의 일관성을 검사하기 위한 규칙을 제안하고, 정의된 일관성 규칙을 적용한 검증 결과는 5장에서 기술된다. 6장에서는 기존 연구와 본 연구와의 차이에 대해서 비교분석을 한다. 끝으로, 7장에서는 본 연구에 대한 토의와 향후 연구 과제에 대해 언급하면서 결론을 내린다.

2. 기본 개념

이 장에서는 기존 특성모델에 대해서 간단히 요약하고, 본 논문에서 정형적으로 정의할 다중 관점의 제품계열아키텍처에 대한 기본 개념을 소개한다.

2.1 특성모델

특성모델은 많은 제품계열 방법에서 제품계열의 공통성과 가변성을 표현하고 관리하기 위해 사용하고 있는 모델로서, 본 논문에서는 특성모델을 정형적으로 정의한 [2]를 기반으로 제품계열아키텍처 모델과의 대응 관계를 설정한다.

특성은 다양한 이해당사자(stakeholder) 관점에서 보여지는 제품간의 구별되는 혹은 두드러진 특징으로 크게 기능 및 비기능 특성으로 구분된다 [3]. 특성모델(feature model)은 제품특성의 집합 F, 구성 규칙의 집합 CR, 특성 간의 여덟 가지 관계(R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8) 로 정의된다 [2]. 즉, 제품특성 집합 F 는 공통특성 집합(F_C), 가변특성 집합(F_V)의 합집합으로 정의되고, 구성 규칙 집합 CR 은 가변특성의 선택을 제한하는 다음 네 가지 종류의 구성규칙 (configuration rule) 들로 구성된다.

- (필수선택) $Req(f_1,f_2)$: 특성 f_1이 선택된다면, 특성 f_2도 반드시 선택되어야 한다
- (배타선택) $Ex(f_1,f_2)$: 두 특성 f_1과 f_2는 동시에 선택되어서는 안 된다.
- (택일선택) $XOR(f, Z)$: 특성 f가 선택된다면, 제품특성 집합 F의 부분집합 Z 내의 특성들 중에 단 하나의 특성만 선택되어야 하고, 특성 f가 선택되지 않는 다면 집합 Z 내의 어떠한 특성도 선택되어서는 안 된다.
- (다중선택) $OR(f, Z)$: 특성 f가 특정한 제품을 위해 선택된다면, 제품특성 집합 F의 부분집합 Z 내의 특성들 중에 적어도 하나의 특성이 반드시 선택되어야 하고, 특성 f가 선택되지 않는 다면 집합 Z내의 어떠한 특성도 선택되어서는 안 된다.

또한, 제품특성 간에는 다양한 관계를 통해 구조화되는데, R_1과 R_2는 기존 특성 모델[3]에서 정의된 집합화와 일반화 관계를 의미하고, R_3에서 R_8까지는 제품특성 간에 존재하는 의존관계를 나타낸다.

위와 같이 정의된 특성모델은 다중 관점의 제품계열아키텍처 모델과의 명시적 대응을 통해서 제품계열아키텍처 모델의 가변성을 관리하는 데 있어 중심적인 역할을 수행한다.

2.2 다중 관점의 제품계열아키텍처

대규모 시스템의 경우, 한 가지 관점으로 아키텍처를 설계하기에는 시스템의 복잡도가 매우 크므로, 관심 분리의 원칙에 입각하여 다중관점으로 아키텍처 설계를 분리하고, 이를 통합하는 것이 효과적이다. 따라서 본 논문에서는 (그림 1)에서 보는 바와 같이, 제품계열아키텍처 설계를 개념 관점, 프로세스 관점, 배치 관점, 모듈 관점으로 분리한다.

개념 관점 설계는 기능적 특성(functional feature)이 어떻게 논리적인 단위의 개념 컴포넌트로 대응되는 지에 초점을 두어 이루어진다. 프로세스 관점의 설계는 논리적인 개념 컴포넌트들 사이의 동시성(concurrency) 구조를 표현하는데 초점을 두고, 배치 관점의 설계는 동시성을 지닌 프로세스 컴포넌트들을 실질적인 자원을 지닌 노드(node) 컴포넌트에 어떻게 배치시킬 지를 결정한다. 주지할 점으로 프로세스 관점과 배치 관점의 설계 결정에 영향을 미치는 주요 요소는 비기능적 특성 (non-functional feature)이다. 마지막으로 모듈 관점의 설계는 개념, 프로세스, 배치 아키텍처 관점 설계에서 내려진 설계 결정이 어떻게 구현 단위인 모듈 컴포넌트로 대응되는 지를 나타낸다.

이와 같은 다중 관점의 아키텍처 설계는 제품계열아키텍처 설계의 고유한 방법은 아니다. Kruchten [4]은 일찍이 소프트웨어 아키텍처를 다양한 이해당사자(stakeholder)의 관심사에 따라 논리 관점(logical view), 개발 관점(development view), 프로세스 관점(process view), 물리적 관점(physical view)로 구분하는 모델을 제안하였다. 본 논문의 개념 관점은 Kruchten의 논리 관점에 해당하고, 배치 관점은 물리적 관점으로, 모듈 관점은 개발 관점으로 대응된다.

또한, 다중 관점의 설계를 제품계열 아키텍처에 적용한 다른 연구도 본 논문이 처음은 아니다. 예를 들면, Thiel et al [5]은 논리 관점, 프로세스 관점, 물리 관점, 배치 관점에서 제품계열아키텍처를 위한 가변성 모델을 제안하였다. 다중 관점의 아키텍처 설계를 제품계열 상황에 적용할 때에 반드시 고려해야 할 점은 각 관점의 제품계열아키텍처가 가지는 가변요소를 모델링 하는 것이 필수적이고, 각 관점의 가변요소간에 일관성이 유지되도록 가변성을 관리해야 한다.

본 논문에서는 가변성 표현을 위한 모델링 언어를 제공하는 기존 연구와는 달리, 각 관점의 제품계열아키텍처 모델이 가지는 가변요소를 집합 이론(set theory)를 이용하여 정형적으로 정의한다. 이는 서로 다른 관점의 제품계열아키텍처 모델의 일관성 규칙을 일차 논리 (first-order logic)를 이용하여 정의하기 위한 기반을 마련한다.

3. 다중 관점의 제품계열아키텍처 모델

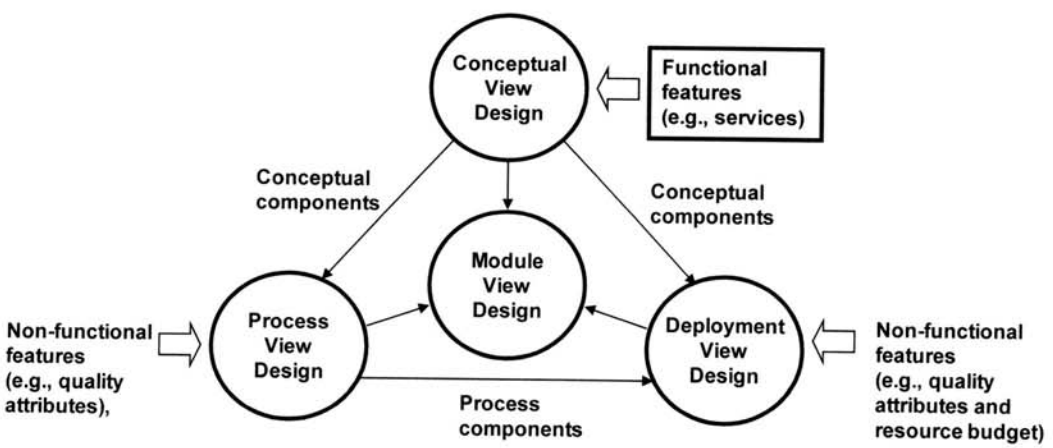
이 장에서는 제품계열아키텍처 모델을 정형적으로 정의하고, 특성모델과 제품계열아키텍처 모델과의 대응 및 서로 다른 관점의 제품계열아키텍처 모델간의 대응에 대해서 논의한다. 특히, 각 관점의 제품계열아키텍처 모델에서 어떠한 아키텍처 가변요소가 있는 지에 초점을 두어 설명한다.

3.1 개념 관점의 제품계열아키텍처 모델

개념 관점의 제품계열아키텍처 모델은 시스템의 논리적인 구성 단위인 개념 컴포넌트와 이들 간의 연결관계를 나타내는 시스템의 구조를 제품계열 환경에서 나타낸 것이다. 일반적으로, 개념 컴포넌트들 간의 복잡한 상호작용은 커넥터를 통해 정의될 수도 있지만, 본 논문에서는 컴포넌트간의 상호작용을 담당하는 커넥터도 일종의 컴포넌트로 정의한다. 그 이유는 본 논문의 요지인 아키텍처 가변성 관리의 논리적인 흐름을 유지하면서 간결한 서술을 위해서이다. 따라서 본 논문에서 개념 관점의 제품계열아키텍처 모델은 다음과 같이 정의된다.

정의 1 개념 관점의 제품계열아키텍처 모델

개념 관점의 제품계열아키텍처 모델은 $CAM = < CCP, CCC >$ 의 두 튜플로 정의된다. CCP 는 개념 컴포넌트의 집합이고, CCC 는 개념 컴포넌트 간의 연결관계를 나타내는 집합이다.



(그림 1) 다중 관점의 제품계열아키텍처 설계

정의 2 개념 컴포넌트(Conceptual Component)

개념 컴포넌트는 $<P, B, F_{CON}, M_{B2P}, M_{B2F}>$의 다섯 튜플로 정의된다. P는 컴포넌트 포트(port)의 집합, B는 컴포넌트의 기능 행위의 집합, F_{CON}는 개념 컴포넌트에 대응된 기능 특성의 집합, M_{B2P}는 행위의 포트 순응 대응, M_{B2F}은 행위의 특성 실현대응을 의미한다.

컴포넌트 포트는 다른 컴포넌트와의 상호작용을 하기 위한 인터페이스 역할을 하며, 컴포넌트의 기능 행위 중에서 다른 컴포넌트와의 상호작용을 담당하는 부분 행위를 나타낸다. 따라서 컴포넌트의 기능행위는 컴포넌트의 포트 행위와 부합되도록 정의되어야 한다.

제품계열 환경에서 개념 컴포넌트는 제품에 따라 포트(인터페이스)의 가변성을 가질 수도 있으므로, 개념 컴포넌트의 컴포넌트 포트 집합(P)은 모든 제품에 공통인 포트집합(CP)과 제품에 따라 선택될 수 있는 가변 포트 집합(VP)의 합집합이다. 또한 개념 컴포넌트가 인터페이스는 동일하지만 제품에 따라 다양한 기능 행위를 가질 수도 있으므로, 제품계열 환경에서 개념 컴포넌트는 하나 이상의 기능 행위를 가질 수 있다.

정의 3 행위의 포트 순응 대응(M_{B2P})

M_{B2P}는 B에서 2^P 로의 대응(M_{B2P}:$B \rightarrow 2^P$)으로서 기능 행위 $b \in B$가 공통 포트 집합(CP)과 가변 포트 집합의 부분 집합 (2^{VP})을 구현한 경우에 대응 관계가 정의된다.

가령, 어떤 개념 컴포넌트에 대해서, $B=\{b1, b2\}$, $CP=\{p1, p2\}$, $VP=\{p3, p4\}$이고, $M_{B2P}=\{(b1, \{p1, p2, p3\}), (b2, \{p1, p2, p4\})\}$ 라면, 기능 행위 $b1$은 공통 포트 $p1$, $p2$와 가변 포트인 $p3$를 구현한 것이고, $b2$는 공통 포트 $p1$, $p2$와 가변 포트 $p4$를 구현한 것을 의미한다.

정의 4 행위의 특성 실현 대응(M_{B2F})

M_{B2F}은 기능 행위 $b \in B$가 개념 컴포넌트에 할당된 특성 집합 (F_{CON}) 중에서 공통특성 집합(CF_{CON})과 가변특성 집합 (VF_{CON})의 부분 집합 ($2^{VF_{CON}}$)을 구현한 경우 $B \rightarrow CF_{CON} \cup 2^{VF_{CON}}$ 의 함수로서 정의된다.

따라서, 특정한 제품을 위한 가변 특성의 선택이 결정된 후에는 각 개념 컴포넌트 마다 유일한 하나의 기능 행위가 결정될 수 있다. 가령 특정 개념 컴포넌트 c에 대해서 $B=\{b1,b2\}$, $CF_{CON}=\{f1\}$, $VF_{CON}=\{f2, f3\}$, $M_{B2F}=\{(b1,\{f1,f2\}), (b2,\{f1,f3\})\}$라고 할 때, 가변특성 $f2$, $f3$ 중에서 $f2$가 선택이 되고, $f3$은 선택이 안된 경우에는 기능행위 $b1$이 개념 컴포넌트의 c의 기능행위로 결정된다.

정의 5 개념 컴포넌트 연결(Conceptual Component Connection)

개념 컴포넌트 연결관계 집합 CCC는 $CCP \times P$에서 $CCP \times P$로의 이진 관계(binary relation)로 정의된다.

- 개념 컴포넌트 c의 개념 포트 집합을 $P(c)$라 하자.
- 임의의 서로 다른 개념 컴포넌트 $s,d \in CCP$와 컴포넌트 포트 $p1 \in P(s)$, $p2 \in P(d)$ 대해서, $((s,p1), (d,p2)) \in CCC \leftrightarrow$ "컴포넌트 포트 p1과 p2 사이의 프로토콜이 일치"

개념 아키텍처 구성은 개념 컴포넌트간의 연결관계를 통해서 이루어진다. 제품계열 환경에서 개념 컴포넌트 자체가 제품에 따라서 선택적으로 포함될 수도 있고, 삭제될 수도 있으므로, 이러한 가변적인 개념 컴포넌트와 다른 개념 컴포넌트 간의 연결관계도 가변적이게 된다. 따라서 개념 컴포넌트간의 연결관계 가변성은 연관된 개념 컴포넌트의 가변성과 동기화된다. 즉 특정 개념 컴포넌트가 특정 제품을 위한 아키텍처 구성에서 포함되거나 삭제된다면, 그와 연관된 다른 개념 컴포넌트와의 연결관계도 같이 포함되거나 삭제된다.

3.2 프로세스 관점의 제품계열아키텍처 모델

프로세스 관점의 아키텍처는 시스템의 구조를 독립적인 실행 단위인 프로세스 컴포넌트들과 이들 사이의 연결 관계로 나타낸다. 따라서 프로세스 관점의 아키텍처를 다음과 같이 정의한다.

정의 6 프로세스 관점의 제품계열아키텍처

프로세스 관점의 제품계열아키텍처 모델은 $PAM=<PCP, M_{p2c}, PCC>$의 세 튜플로 정의된다. PCP는 프로세스 컴포넌트의 집합, M_{p2c}는 개념 컴포넌트와 프로세스 컴포넌트와의 대응관계, PCC는 두 프로세스 컴포넌트간의 연결관계를 나타내는 집합이다.

프로세스 컴포넌트는 독립적으로 수행될 수 있는 소프트웨어 단위로서, 비기능 특성에 따라서 개념 컴포넌트가 서로 다른 프로세스 컴포넌트에 할당 될 수 있다. 따라서 프로세스 컴포넌트와 개념 컴포넌트와의 대응관계는 다음과 같이 정의된다.

정의 7 프로세스 컴포넌트와 개념 컴포넌트의 대응

M_{p2c}는 PCP에서 $2^{CCP} \times 2^F$로의 대응(M_{p2c}: $PCP \rightarrow 2^{CCP} \times 2^F$)으로서, 개념 아키텍처의 개념 컴포넌트 집합 (CCP)의 부분집합인 $C \in 2^{CCP}$가 특성 모델의 제품특성 집합(F)의 부분집합인 $FS \in 2^F$가 선택되었을 때 프로세스 아키텍처의 특정 프로세스 컴포넌트 $p \in PCP$에 할당되는 경우에 정의된다. 즉, $M_{p2c}(p)=(C,FS)$ 혹은 $(p, (C,FS)) \in M_{p2c}$

예를 들면, 성능(performance)이 중요한 제품에서는 동일한 주기로 실행될 수 있는 두 개의 병렬적인 개념 컴포넌트 $c1$, $c2$를 하나의 프로세스 컴포넌트 $p1$에 할당시켜, 프로세스들의 스케듈링 오버헤드 및 프로세스 간의 통신 오버헤드를 줄임으로써 전반적인 성능을 향상시킬 수 있지만, 가용성(availability)이 중요한 제품에서는 병렬적인 두 개념 컴포넌트를 독립적인 프로세스 컴포넌트 $p2$, $p3$에 할당하여 수

행시킴으로써, 둘 중 한 프로세스 컴포넌트의 오류가 다른 프로세스 컴포넌트의 수행에 영향을 주지 않도록 하는 것이 좋을 수 있다. 따라서, 비기능 특성인 성능을 P, 가용성을 A라고 가정하면, 위의 프로세스 컴포넌트와 개념 컴포넌트 간에는 M_{p2c}={(*p1*, ({*c1*,*c2*},{*P*})), (*p2*, ({*c1*},{*A*})), (*p3*, ({*c2*},{*A*}))}의 대응 관계가 정의된다.

정의 8 프로세스 컴포넌트 연결(Process Component Connection)
프로세스 컴포넌트 연결관계 집합 PCC는 PCP에서 PCP로의 이진 관계(binary relation)으로 다음과 같이 정의된다.

- 임의의 서로 다른 프로세스 컴포넌트 $s,d \in PCP$에 대해서, $(s,d) \in PCC \leftrightarrow$ "s는 d와 연결되어 통신"

프로세스 관점의 아키텍처 구성은 프로세스 컴포넌트 간의 연결관계를 통해서 이루어진다. 제품계열 환경에서 프로세스 컴포넌트도 개념 컴포넌트와 마찬가지로 특정 제품을 위한 아키텍처에서는 포함될 수도 있고, 포함되지 않을 수도 있다. 가령, 앞의 예에서 프로세스 컴포넌트 *p1*, *p2*, *p3*는 P(성능) 혹은 A(가용성) 특성의 선택 여부에 따라서 포함될 수도 있고, 그렇지 않을 수도 있다. 즉, P 가 특정 제품을 위해서 선택되었다면, 그 제품을 위한 프로세스 관점의 아키텍처는 개념 컴포넌트 *c1*과c2가 할당된 프로세스 컴포넌트 *p1*을 포함하게 될 것이다. 따라서, 특정 제품을 위한 프로세스 관점의 아키텍처는 그 제품을 위해 포함된 프로세스 컴포넌트들과 그들간의 관계로 구성된다.

3.3 배치 관점의 제품계열아키텍처 모델

배치 관점의 아키텍처는 자원을 가진 물리적인 개체인 노드 컴포넌트들과 이들 사이의 연결관계를 나타낸다. 따라서 배치 관점의 아키텍처도 프로세스 관점의 아키텍처와 유사하게 정의된다. .

정의 9 배치 관점의 제품계열아키텍처
배치 관점의 제품계열아키텍처 모델은 DAM=<NCP, M_{n2p}, NCC>의 세 튜플로 정의된다. NCP 는 노드 컴포넌트의 집합, M_{n2p} 는 프로세스 컴포넌트와 노드 컴포넌트와의 대응관계, NCC는 두 노드 컴포넌트간의 연결관계를 나타내는 집합이다.

제품계열 환경에서 프로세스 컴포넌트는 비기능적 특성에 따라서, 서로 다른 노드 컴포넌트에 할당될 수 있다. 가령, 성능이 중요한 제품에서는 병렬처리를 위해서 프로세스 컴포넌트를 각기 다른 노드 컴포넌트에 할당 시킬 수도 있고, 비용이 중요한 제품에서는 이들을 하나의 노드 컴포넌트에 할당 시킬 수도 있다. 따라서 노드 컴포넌트와 프로세스 컴포넌트간의 대응관계는 다음과 같이 정의된다.

정의 10 노드 컴포넌트와 프로세스 컴포넌트의 대응
M_{n2p}는 NCP에서 $2^{PCP} \times 2^{F}$로의 대응(M_{n2p}: $NCP \rightarrow 2^{PCP} \times 2^{F}$)으로서, 프로세스 아키텍처의 프로세스 컴포넌트 집합(PCP)의 부분집합 $C \in 2^{PCP}$가 특성모델의 제품특성 집합 (F)의 부분집합 $FS \in 2^{F}$가 선택되었을 때, 배치 아키텍처의 노드 컴포넌트 $n \in NCP$에 할당되는 경우에 정의된다. 즉, M_{n2p} (n) = (C,FS) 혹은 (n, (C,FS)) $\in M_{n2p}$

정의 11 노드 컴포넌트 연결(Node Component Connection)
노드 컴포넌트 연결관계 집합 NCC는 NCP 에서 NCP로의 이진 관계(binary relation) 로 다음과 같이 정의된다.

- 임의의 서로 다른 노드 컴포넌트 $s,d \in NCP$에 대해서, $(s,d) \in NCC \leftrightarrow$ "s는 d와 물리적인 링크로 연결"

배치 아키텍처 구성은 노드 컴포넌트간의 연결관계를 통해 나타난다. 즉, 노드 컴포넌트들 간의 연결을 통해서 배치 아키텍처가 구성된다. 배치 아키텍처의 경우에도 프로세스 아키텍처와 마찬가지로 노드 컴포넌트가 제품에 따라서 포함될 수도 있고 포함되지 않을 수도 있다. 그러므로, 특정 제품을 위한 배치 아키텍처는 그 제품을 위해 포함된 노드 컴포넌트들과 그들 간의 연결 관계를 통해 구성된다.

3.4 모듈 관점의 제품계열아키텍처 모델

모듈 관점의 아키텍처는 개념, 프로세스, 배치 관점의 아키텍처에서 내려진 아키텍처 설계 결정이 어떻게 구현 단위인 모듈로 대응되는 가를 나타낸다. 모듈 간에는 사용(Usage) 관계를 통해서 연결이 이루어진다.

정의 12 모듈 관점의 제품계열아키텍처
모듈 관점의 제품계열아키텍처 모델은 MAM =< MCP, MUR >의 두 튜플로 정의된다. MCP는 모듈 컴포넌트의 집합, MUR은 두 모듈 컴포넌트간의 사용관계를 나타내는 집합이다.

정의 13 모듈 컴포넌트 (Module Component)
모듈 컴포넌트는 <$PI \cup RI$, IMP, F_{MOD}, M_{IM2I}, M_{IM2F}> 의 다섯 튜플로 정의된다. PI 는 모듈 컴포넌트가 다른 모듈 컴포넌트에 제공하는 인터페이스의 집합, RI 는 모듈 컴포넌트가 요구하는 다른 컴포넌트의 인터페이스 집합, IMP 는 모듈 컴포넌트의 구현 집합, F_{MOD} 는 모듈 컴포넌트에 대응된 특성의 집합, M_{IM2I},는 모듈 인터페이스 순응 대응, M_{IM2F} 는 모듈 특성 실현 대응을 의미한다.

모듈 컴포넌트의 구현($imp \in IMP$)은 요구되는 다른 모듈 컴포넌트의 인터페이스(RI)를 활용하여 모듈 컴포넌트가 제공하는 인터페이스(PI)를 모두 실현해야 한다. 제품계열 환경에서는 동일한 인터페이스를 실현하는 여러 구현이 있을 수 있으므로 서로 다른 구현이 동일한 인터페이스 집합과 대응관계를 가질 수도 있고, 모듈 컴포넌트의 인터페이스가 제품에 따라 변할 수도 있으므로 모듈 컴포넌트의 구현은 모듈 컴포넌트가 제공하는 모든 인터페이스를 반드시 구현

할 필요 없이, 인터페이스의 부분집합을 구현할 수도 있다. 따라서, 모듈 컴포넌트의 구현과 모듈 인터페이스 간에는 다음과 같은 대응관계를 정의할 수 있다.

정의 14 모듈 구현의 인터페이스 순응 대응 (M_{IM2I})
M_{IM2I}는 $IMP \rightarrow 2^{PI \cup RI}$의 함수로서, 모듈 구현 $imp \in IMP$가 인터페이스 부분 집합 $i \in 2^{PI \cup RI}$를 구현한 경우에 정의된다.

모듈 컴포넌트의 각 구현과 모듈 컴포넌트에 할당된 특성 간의 대응관계는 다음과 같이 정의된다.

정의 15 모듈 구현의 특성 실현 대응 (M_{IM2F})
M_{IM2F}는 $IMP \rightarrow 2^{VF_{MOD}}$의 함수로서, 모듈 구현 $imp \in IMP$가 컴포넌트에 할당된 특성 집합 (F_{MOD}) 중에서 공통특성 집합(CF_{MOD})과 가변특성 집합(VF_{MOD})의 부분 집합 ($2^{VF_{MOD}}$)을 구현한 경우 $IMP \rightarrow CF_{MOD} \cup 2^{VF_{MOD}}$의 함수로서 정의된다.

모듈들은 그들이 제공하는 인터페이스와 요구되는 인터페이스가 서로 일치하는 경우에 사용 관계에 의해서 연결이 이루어진다. 따라서 모듈 컴포넌트간의 사용관계 집합인 MUR는 다음과 같이 정의된다.

정의 16 모듈 사용 관계 (Module Usage Relation)
모듈 사용 관계 집합 MUR은 $MCP \times (PI \cup RI)$에서 $MCP \times (PI \cup RI)$로의 이진 관계(binary relation)으로 정의된다.

- 모듈 컴포넌트 m의 요구 인터페이스 집합과 제공 인터페이스 집합을 각각 $RI(m)$과 $PI(m)$이라 하자.
- 임의의 서로 다른 모듈 컴포넌트 $s,d \in MCP$와 컴포넌트 인터페이스 $i1 \in RI(s)$, $i2 \in PI(d)$ 대해서, $((s,i1),(d,i2)) \in MUR \leftrightarrow i1=i2$

모듈 관점의 아키텍처와 개념 관점의 아키텍처 정의가 매우 비슷하지만, 모듈 컴포넌트의 단위는 개념 관점의 아키텍처뿐만 아니라 프로세스 및 배치 관점의 아키텍처에서 내려진 설계 결정을 반영한 구현의 단위라는 측면에서 개념 컴포넌트의 단위랑 일치하지 않을 수 있다.

요약하면, 이상의 제품계열아키텍처 모델은 특성모델과 명시적인 대응관계를 포함하고 있을 뿐만 아니라, 각 관점의 제품계열아키텍처 모델 간에도 대응관계가 정의되어 있다. 이러한 대응은 아키텍처 설계 시에 설계자에 의해서 정의되는 것이므로, 대응이 불완전하거나 일관성이 결여될 가능성이 있다. 다음 장에서는 정형적으로 정의된 제품계열아키텍처 모델을 바탕으로 제품계열아키텍처 모델 안에 정의된 다양한 대응 관계 사이에 일관성을 검사하기 위한 규칙을 정의한다.

4. 제품계열아키텍처 모델의 일관성

본 논문에서는 제품계열아키텍처 모델의 일관성을 크게 세 가지 관점에서 정의한다. 첫째, 각 관점의 제품계열아키텍처 모델은 그 자체 구성요소간에 일관성이 있어야 한다. 둘째, 서로 다른 관점의 제품계열아키텍처 모델 간에 일관성이 있어야 한다. 셋째, 특성 모델로 표현된 제품계열의 가변성과 제품계열아키텍처 모델의 가변성 간의 일관성이 있어야 한다. 따라서 이 장에서는 이들 각각에 대한 구체적인 일관성 규칙을 정형적으로 정의한다.

구체적인 일관성 규칙을 정의하기에 앞서, 일관성 규칙 정의에서 사용되는 유용한 함수 및 집합은 다음과 같다.

- $DConnected_{CCP}(c1, c2)$: 두 개념 컴포넌트 $c1$, $c2$가 포트를 통해서 서로 직접 연결되어 있으면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$DConnected_{CCP}(c1,c2) \equiv \exists_{x \in P(c1)} \exists_{y \in P(c2)} \bullet ((c1,x),(c2,y)) \in CCC$$

- $IConnected_{CCP}(c1, c2)$: 두 개념 컴포넌트 $c1$과 $c2$사이에 간접적인 연결관계가 성립하면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$IConnected_{CCP}(c1,c2) \equiv \exists_{n \in N} \bullet C_C^n(c1) = c2$$

여기서 $C_C^n(x)$는 개념 컴포넌트 x로부터 길이 n의 패스로 연결된 개념 컴포넌트 집합을 반환하는 함수이고, 이는 개념 컴포넌트 x와 직접적인 연결관계에 있는 개념 컴포넌트 집합을 반환하는 함수 $C_C(x)$의 n번 합성으로 정의된다.

$$C_C^{\,n}(c) \equiv C_C \circ \ldots \circ C_C(c) \quad (\circ : \text{function composition})$$
$$C_C(c) \equiv \{x \mid DConnectedCCP(c,x)\}$$

- $Connected_{CCP}(c1, c2)$: 두 개념 컴포넌트 $c1$, $c2$가 직접 혹은 간접적으로 서로 연결되어 있으면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$Connected_{CCP}(c1,c2) \equiv DConnected_{CCP}(c1,c2) \vee IConnected_{CCP}(c1,c2)$$

- $DConnected_{MCP}(m1, m2)$: 두 모듈 컴포넌트 $m1$, $m2$가 모듈 사용 관계를 통해 연결되어 있으면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$DConnected_{MCP}(m1,m2) \equiv \exists_{x \in RI(m1)} \exists_{y \in PI(m2)} \bullet ((m1,x),(m2,y)) \in MUR$$

- $IConnected_{MCP}(m1, m2)$: 두 모듈 컴포넌트 $m1$, $m2$ 사이에 간접적인 연결관계가 성립하면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$IConnected_{MCP}(m1,m2) \equiv \exists_{n \in N} \bullet C_M^{\,n}(m1) = m2$$

여기서 $C_M^n(x)$는 $C_C^n(x)$와 유사하게 모듈 컴포넌트 x로부

터 길이 n의 패스로 연결된 모듈 컴포넌트를 반환하는 함수이다.

- $Connected_{MCP}(m1, m2)$: 두 모듈 컴포넌트 $m1$, $m2$ 가 직접 혹은 간접적으로 서로 연결되어 있으면 참을 반환, 그렇지 않으면 거짓을 반환하는 함수

$$Connected_{MCP}(m1, m2) \equiv DConnected_{MCP}(m1, m2) \vee IConnected_{MCP}(m1, m2)$$

- $Mutex(f1,f2)$: 두 특성 $f1$과 $f2$가 같은 제품을 위해서 선택될 수 없을 때, 참을 반환하는 함수 이다. 두 특성이 같이 선택될 수 없는 경우는 두 특성 사이에 배타선택 혹은 택일선택의 구성 규칙이 정의된 경우이므로, 다음과 같이 정의된다.

$$Mutex(f1, f2) = \left(Ex(f1, f2) \in CR\right) \vee \left(Ex(f2, f1) \vee \exists_{XOR(z,S) \in CR} \bullet S \supseteq \{f1, f2\}\right)$$

단, CR은 특성모델에 정의된 구성 규칙의 집합을 의미한다 (2.1절 참조).

- $P(c)$: 개념 컴포넌트 c의 포트 집합
- $B(c)$: 개념 컴포넌트 c의 행위 집합
- $F_{CON}(c)$:개념 컴포넌트 c에 대응된 특성 집합
- $PI(m)$: 모듈 컴포넌트 m의 제공 인터페이스 집합
- $RI(m)$: 모듈 컴포넌트 m의 요구 인터페이스 집합
- $IMP(m)$: 모듈 컴포넌트 m의 구현 집합
- $F_{MOD}(m)$:개념 컴포넌트 m에 대응된 특성 집합
- $MappedCCP(p)$: 프로세스 컴포넌트 p에 대응된 개념 컴포넌트의 집합

$$MappedCCP(p) \equiv \{c \in CCP \mid M_{p2c}(p) = (c, f) \wedge f \in F\}$$

- $F_{PRO}(p)$:프로세스 컴포넌트 p에 직접 대응된 특성 집합과 프로세스 컴포넌트 p에 할당된 개념 컴포넌트들에 대응된 특성 집합의 합집합

$$F_{PRO}(p) \equiv \{f \in F \mid M_{p2c}(p) = (c, f) \wedge c \in MappedCCP(p)\} \cup \{F_{CON}(c) \mid c \in MappedCCP(p)\}$$

- $MappedPCP(n)$: 노드 컴포넌트 n에 대응된 프로세스 컴포넌트의 집합

$$MappedPCP(n) \equiv \{p \in PCP \mid \exists_{f \in FS} \bullet M_{n2p}(n) = (p, f)\}$$

- $F_{NOD}(n)$:노드 컴포넌트 n에 직접 대응된 특성 집합과 노드 컴포넌트 n에 할당된 프로세스 컴포넌트들에 대응된 특성 집합의 합집합

$$F_{NOD}(n) \equiv \{f \in F \mid M_{n2p}(n) = (p, f) \wedge p \in MappedPCP(n)\} \cup \{F_{PRO}(p) \mid p \in MappedPCP(n)\}$$

4.1 아키텍처 모델 구성요소 간의 일관성

각 관점의 제품계열아키텍처 모델은 누락된 구성요소가 없어야 하고, 모델 구성요소간에 일관성이 있어야 한다.

규칙 1 개념 아키텍처의 일관성

1. 개념 아키텍처에서 서로 다른 개념 컴포넌트가 둘 이상 존재할 때, 모든 개념 컴포넌트는 그와 연결된 개념 컴포넌트가 적어도 하나는 있어야 한다.

$$|CCP| \geq 2 \rightarrow \forall_{c1 \in CCP} \exists_{c2 \in CCP} \bullet c1 \neq c2 \wedge DConnected_{CCP}(c1, c2)$$

2. 개념 컴포넌트의 포트 중에 다른 개념 컴포넌트의 포트와 연결되지 않는 것은 없어야 한다. 즉 임의의 개념 컴포넌트 $c1$에 대해서, $c1$의 모든 포트들은 $c1$과 다른 어떤 개념 컴포넌트 $c2$의 한 포트 $p2$와 연결관계가 성립해야 한다.

$$\forall_{c1 \in CCP} \forall_{p1 \in P(c1)} \exists_{c2 \in CCP} \exists_{p2 \in P(c2)} \bullet c1 \neq c2 \wedge \left((c1, p1), (c2, p2)\right) \in CCC$$

3. 개념 컴포넌트의 포트 중에 개념 컴포넌트의 행위에 의해서 대응되지 않는 것이 없어야 한다. 즉, 임의의 개념 컴포넌트 c에 대해서, c의 모든 포트들은 c의 어떤 행위와 대응관계를 가져야 한다.

$$\forall_{c \in CCP} \forall_{p \in P(c)} \exists_{b \in B(c)} \bullet p \in M_{B2P}(b)$$

규칙 2 프로세스 및 배치 아키텍처의 일관성

1. 프로세스 아키텍처에서 서로 다른 프로세스 컴포넌트가 둘 이상 존재할 때, 모든 프로세스 컴포넌트는 그와 연결된 프로세스 컴포넌트가 적어도 하나는 있어야 한다.

$$|PCP| \geq 2 \rightarrow \forall_{p1 \in PCP} \exists_{p2 \in PCP} \bullet p1 \neq p2 \wedge (p1, p2) \in PCC$$

2. 배치 아키텍처에서 서로 다른 노드 컴포넌트가 둘 이상 존재할 때, 모든 노드 컴포넌트는 그와 연결된 노드 컴포넌트가 적어도 하나는 있어야 한다

$$|NCP| \geq 2 \rightarrow \forall_{n1 \in NCP} \exists_{n2 \in NCP} \bullet n1 \neq n2 \wedge (n1, n2) \in NCC$$

규칙 3 모듈 아키텍처의 일관성

1. 모듈 아키텍처에서 서로 다른 컴포넌트가 둘 이상 존재할 때, 모든 컴포넌트는 그와 연결된 컴포넌트가 적어도 하나는 있어야 한다.

$$|MCP| \geq 2 \rightarrow \forall_{c1 \in MCP} \exists_{c2 \in MCP} \bullet c1 \neq c2 \wedge DConnected_{MCP}(c1, c2)$$

2. 모듈 컴포넌트의 요구 인터페이스 중에 다른 모듈 컴포넌트의 제공 인터페이스와 연결되지 않는 것은 없어

야 한다. 즉, 임의의 모듈 컴포넌트 *m1*에 대해서, *m1*의 모든 요구 인터페이스는 다른 어떤 모듈 컴포넌트 *m1*의 한 제공 인터페이스 *i2*와 연결관계가 성립해야 한다.

$$\forall_{m1\in MCP}\forall_{i1\in RI(m1)}\exists_{m2\in MCP}\exists_{i2\in PI(m2)}\bullet m1\neq m2\wedge((m1,i1),(m2,i2))\in MUR$$

3. 모듈 컴포넌트의 제공 혹은 요구 인터페이스 중에 모듈 컴포넌트의 구현과 대응되지 않는 것이 없어야 한다. 즉, 임의의 모듈 컴포넌트 m에 대해서, m의 모든 인터페이스 집합들은 m의 어떤 구현 명세와 대응관계를 가져야 한다.

$$\forall_{m\in MCP}\forall_{interface\in RI(m)\cup PI(m)}\exists_{imp\in IMP(m)}\bullet interface\in M_{IM21}(imp)$$

4.2 다중 관점 아키텍처 간의 일관성

각 관점의 제품계열아키텍처 모델간에는 서로 대응관계가 존재한다. 이러한 대응관계 사이에는 일관된 규칙이 존재해야 한다.

규칙 4. 개념 관점과 프로세스 관점의 일관성

1. 모든 개념 컴포넌트는 대응 되는 프로세스 컴포넌트가 반드시 하나 이상 존재해야 한다.

$$\forall_{c\in CCP}\exists_{p\in PCP}\bullet c\in MappedCCP(p)$$

2. 서로 다른 프로세스 컴포넌트 간에 연결관계가 존재한다면, 이들 프로세스 컴포넌트에 할당된 개념 컴포넌트 사이에 연결관계가 존재해야 한다.

$$\forall_{(pc1,pc2)\in PCC}\exists_{cc1\in MappedCCP(pp1)}\exists_{cc2\in MappedCCP(pp2)}\bullet Connected_{CCP}(cc1,cc2)$$

규칙 5 프로세스 관점과 배치 관점의 일관성

서로 다른 노드 컴포넌트 간에 연결관계가 존재한다면, 이들 노드 컴포넌트에 할당된 프로세스 컴포넌트 사이에 연결관계가 존재해야 한다.

$$\forall_{(nc1,nc2)\in NCC}\exists_{pc1\in MappedPCP(nc1)}\exists_{pc2\in MappedPCP(nc2)}\bullet(pc1,pc2)\in PCC$$

규칙 6 모듈 관점과 개념, 프로세스, 배치 관점의 일관성

모듈 관점 아키텍처는 개념, 프로세스, 배치 관점의 아키텍처 설계 결정을 모두 반영해야 하므로, 모듈 관점 아키텍처가 구현하는 특성은 개념, 프로세스, 그리고 배치 관점의 아키텍처가 구현하는 특성과 일치해야 한다.

$$\bigcup_{c\in CCP}F_{CON}(c)\cup\bigcup_{p\in PCP}F_{PRO}(p)\cup\bigcup_{n\in NCP}F_{NOD}(n)=\bigcup_{m\in MCP}F_{MOD}(m)$$

4.3 특성모델과 제품계열아키텍처 모델의 일관성

제품계열아키텍처 모델은 특성모델을 아키텍처 관점에서 실현시킨 것이다. 따라서 특성모델과 제품계열아키텍처 모델 사이에는 일관성이 있어야 한다.

규칙 7 개념 컴포넌트와 특성 대응의 일관성

하나의 제품에 같이 선택될 수 없는 특성들은 동일한 행위 명세에 대응되어서는 안된다.

$$\forall_{c\in CCP}\exists_{f1,f2\in F_{CON}(c)}\bullet\left(Mutex(f1,f2)\rightarrow\neg\exists_{b\in B(c)}\bullet M_{B2P}(b)\supseteq\{f1,f2\}\right)$$

앞서 설명했듯이, 제품계열 환경에서 하나의 개념 컴포넌트는 여러 개의 행위 명세를 포함할 수 있다. 하지만, 특정 제품을 위해서는 하나의 행위 명세만 개념 컴포넌트에 바인딩이 이루어져야 한다. 특정한 행위 명세가 개념 컴포넌트와 바인딩이 이루어지려면, 행위 명세와 대응되는 모든 특성이 선택되어야 한다. 따라서 하나의 제품에 같이 존재할 수 없는 특성들은 같은 행위 명세와 대응관계를 이루어서는 안 된다. 모듈컴포넌트와 특성 대응관계도 이와 유사하다.

규칙 8 모듈 컴포넌트와 특성 대응의 일관성

하나의 제품에 같이 선택될 수 없는 특성들은 모듈 컴포넌트의 동일한 구현부에 대응되어서는 안된다.

$$\forall_{m\in MCP}\exists_{f1,f2\in F_{MOD}(m)}\bullet\left(Mutex(f1,f2)\rightarrow\neg\exists_{imp\in IMP(m)}\bullet M_{IM2F}(imp)\supseteq\{f1,f2\}\right)$$

규칙 9 특성 관계와 컴포넌트 연결의 일관성

서로 다른 아키텍처 컴포넌트에 대응된 특성간에 관계가 존재하면 아키텍처 컴포넌트 간에도 직접 혹은 간접적인 연결 관계가 있어야 한다.

1. 개념 관점 아키텍처:

$$\forall_{f1,f2\in\bigcup_i R_i}\bullet\exists_{c1,c2\in CCP}\bullet c1\neq c2\wedge f1\in F_{CON}(c1)\wedge f2\in F_{CON}(c2)\rightarrow Connected_{CCP}(c1,c2)$$

2. 프로세스 아키텍처:

$$\forall_{f1,f2\in\bigcup_i R_i}\bullet\exists_{p1,p2\in PCP}\bullet p1\neq p2\wedge f1\in F_{PRO}(p1)\wedge f2\in F_{PRO}(p2)\rightarrow(p1,p2)\in PCC$$

3. 배치 아키텍처:

$$\forall_{f1,f2\in\bigcup_i R_i}\bullet\exists_{n1,n2\in NCP}\bullet n1\neq n2\wedge f1\in F_{NOD}(n1)\wedge f2\in F_{NOD}(n2)\rightarrow(n1,n2)\in NCC$$

4. 모듈 아키텍처:

$$\forall_{f1,f2\in\bigcup_i R_i}\bullet\exists_{m1,m2\in MCP}\bullet m1\neq m2\wedge f1\in F_{MOD}(m1)\wedge f2\in F_{MOD}(m2)\rightarrow Connected_{MCP}(m1,m2)$$

규칙 10 동시 활성화(concurrent activation) 관계와 프로세스 컴포넌트의 일관성

동시 활성화 관계에 있는 특성은 같은 프로세스 컴포넌트

에 대응되어서는 안 된다.

$$\forall_{f1,f2\in F}\bullet(f1,f2)\in R_7 \wedge \exists_{p\in PCP}\bullet F_{PRO}(p)$$

(R_7은 특성모델에 정의된 특성 간의 관계 중에 동시 활성화 의존성을 나타낸다[2].)

규칙 11 제품계열 일관성

제품계열 아키텍처의 각 관점에 할당된 특성의 합집합은 제품계열 특성의 집합과 같아야 한다.

$$F=\bigcup_{c\in CCP}F_{CON}(c)\cup\bigcup_{p\in PCP}F_{PRO}(p)\cup\bigcup_{n\in NCP}F_{NOD}(n)=\bigcup_{m\in MCP}F_{MOD}(m)$$

이상과 같은 일관성 규칙은 제품계열아키텍처 모델로부터 유효한 제품 아키텍처 모델을 사례화 할 수 있기 위해 지켜져야 하는 선행 조건이다.

5. 일관성 규칙을 활용한 제품계열 아키텍처 모델의 검증

본 장에서는 본 논문에서 제안한 제품계열 아키텍처 모델의 정의와 일관성 규칙의 검증과 타당성을 평가하기 위해서, 엘리베이터 시스템에 대한 사례연구를 수행 결과를 기술한다. 특히, First-Order Logic 기반의 자동화된 모델링 및 검증 도구인Alloy Analyzer [11]를 활용하여 검증을 수행하였다.

5.1 제품계열 특성 모델링

Alloy의 모델링 언어는 객체지향 패러다임을 지원하며 관계(Relation)을 기반으로 한다. 다음 코드 명세는 Alloy 모델링 언어를 이용하여 추상적 특성 모델인 *FeatureModel*을 정의하고, 이를 확장하여 엘리베이터 제품계열을 위한 특성 모델인 *ElevatorPLFeatureModel*에 대한 정의를 나타낸다.

```
1.  abstract sig FeatureModel {
2.          F: set Feature,
3.          R1: Feature -> Feature,
4.          ...
5.          XOR: Feature -> set VFeature,
6.          Req: Feature -> Feature
7.  }
8.  sig ElevatorPLFeatureModel extends FeatureModel{}
9.  {
10.         // the set of features in the elevator product line
11.         F = ElevatorPL + ECSService + Passenger + VIP +...
12.          ...
13.         // Aggregation relationship (R1) between features
14.         ECSService.R1 = Passenger
15.         ECSService.R1 = VIP
16.         ...
17.         // Required configuration between AntiNuisance and WeightSensor
18.         AntiNuisance.Req = WeightSensor
19.         ...
20.}
```

*FeatureModel*은 추상적인 시그니처로 정의되며, 위에서 보는 바와 같이 *F*, *R1*, *XOR*, *Req* 등의 필드를 가진다. *F*는 *Feature* 들의 집합이고, *R1*은 *Feature*간의 이진 관계로 정의된다. *FeatureModel*에는 *R1*이외에도 *R2*~*R8*의 관계도 정의되는데, 이는 지면상 생략하였으나 *R1*과 유사하게 정의된다. 또한, 2장에서 언급한 *XOR*, *OR*, *Req*, *EX* 등의 구성 규칙에 대해서도 이진 관계로 정의된다.

*ElevatorPLFeatureModel*은 엘리베이터 제품계열에 대한 특성 모델로서, *F* 를 엘리베이터 특성들의 집합으로 정의하고 (11 번째 줄), *ECSService*와 *Passenger* 특성 사이와 *ECSService*와 *VIP*특성 사이의 *R1* 관계를 (14, 15 번째 줄), AntiNuisance와 WeightSensor 특성 간의 *Req* 구성 규칙을 (18 번째 줄) 정의한 것이다.

이와 같이 엘리베이터 제품계열의 특성 모델을 Alloy 모델을 이용하여 정의함으로써, 특성 모델 자체에 대한 일관성 검증을 수행할 수가 있을 뿐만 아니라, 다음 절에서 설명할 제품계열 아키텍처 모델과의 일관성 검증을 위한 기반을 마련하게 된다. 특성 모델 자체에 대한 일관성 검증은 본 논문의 범위에서 벗어나므로 언급하지는 않지만, 다음 절에서 언급할 아키텍처 모델의 일관성 검증과 동일한 방식을 취한다.

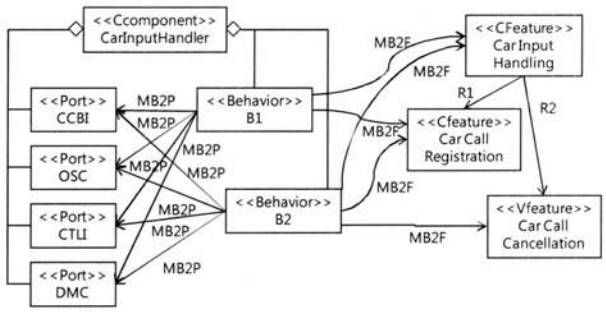
5.2 제품계열 개념 아키텍처 모델링 및 검증

제품계열 개념 아키텍처(정의 1)와 개념 컴포넌트(정의 2)를 Alloy 모델로 표현하면, 다음 코드 명세와 같다.

```
1.  abstract sig CComponent {
2.          P: set Port,
3.          B: set Behavior,
4.          Fcon: set Feature,
5.          MB2P: Behavior -> set Port,
6.          MB2F: Behavior -> set Feature
7.  }
8.  abstract sig ConceptualArchitecture {
9.          CCP: set CComponent,
10.         CCC: (CComponent->Port) -> (CComponent->Port)
11.}
```

*CComponent*와 *ConceptualArchitecture*는 개념 관점의 여러 제품계열 아키텍처를 모델링 할 때 재사용할 수 있는 정의를 나타내며, 특정한 제품계열 개념 아키텍처를 정의하기 위해서는 이들을 확장하여 정의하면 된다.

(그림 2)는 *CarInputHandler* 개념 컴포넌트에 대한 정의를 Alloy 모델을 이용하여 표현한 것(그림 하단)과 UML 모델로 표현한 것(그림 상단)을 보여준다. *CarInputHandler* 컴포넌트는 엘리베이터의 카 내부의 버튼 입력을 처리하는 것으로, 네 개의 포트 (*CCBI*, *OSC*, *CTLI*, *DMC*)와 두 개의 행위 명세(*B1*, *B2*)를 포함하고 있다 (3-4 번째 줄). 이들 포트와 행위 명세 간에는 *행위의 포트 순응 대응* (정의 3) 관계가 *MB2P*의 이진 관계로 정의된다. 즉 행위 *B1*과 *B2* 각각은 네 개의 포트와 *MB2P* 관계로 맺어진다 (6-7 번째 줄). 또한, *CarInputHandler* 개념 컴포넌트에 대응된 특성 집합은 *CarInputHandling*, *CarCallRegistration*, *CarCallCancellation*의 특성들을 포함하며 (5번째 줄), 행위 *B1*과 *B2* 는 *행위의 특성 실현 대응* (정의4) 에 의해서 이들 특성과 *MB2F*의 이진 관계로 맺어진다 (9-11번째 줄). 이의 의미는 행위 *B1*은 특성 *CarInputHandling*과 *CarCallRegistration*을 실현한




```
1. sig CarInputHandler extends CComponent {}
2. {
3.          P =DMC+CCBI+OSC+CTLI
4.          B = B1 + B2
5.          Fcon =CarInputHandling+CarCallRegistration+CarCallCancellation
6.          B1.MB2P=DMC+CCBI+OSC+CTLI
7.          B2.MB2P= DMC+CCBI+OSC+CTLI
8.
9.          B1.MB2F = CarInputHandling+CarCallRegistration
10.         B2.MB2F = CarInputHandling+CarCallRegistration+
11.                   CarCallCancellation
12.}
```

(그림 2) CarInputHandler 개념 컴포넌트의 정의

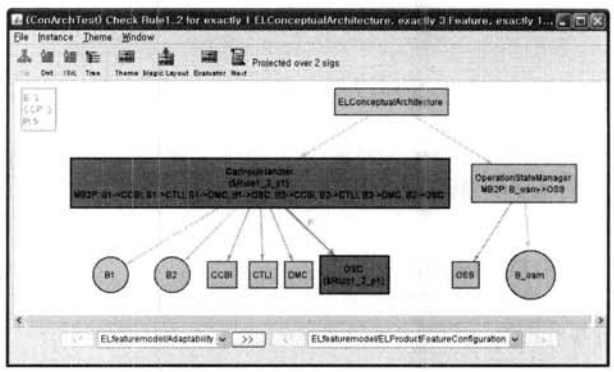
것이고, 행위 *B2*는 *CarInputHandling*, *CarCallRegistration*, *CarCallCancellation*을 실현한 것으로, 가변 특성인 *CarCallCancellation*의 선택 여부에 따라서 *B1*와 *B2* 둘 중에 하나가 *CarInputHandler* 컴포넌트의 행위 명세로 결정됨을 나타낸다.

다음 코드 명세는, 엘리베이터 제품계열의 개념 아키텍처인 *ELConceptualArchitecture*를 정의한 것으로서, *CCP*를 엘리베이터 제품계열을 위한 개념 컴포넌트들의 집합(*CarInputHandler*와 *CallDataManager*포함)으로 정의하고 (3번째 줄), *CCC*를 *CCP*에 정의된 개념 컴포넌트 간의 연결 관계로 설정함으로써 정의된다. 가령, 6-9번째 줄은 *CarInputHandler*와 *CallDataManager* 컴포넌트들이 *DMC*와 *DMS* 포트를 통해서 서로 연결된 것을 나타낸다.

```
1. sig ELConceptualArchitecture extends ConceptualArchitecture{}
2. {
3.         CCP = CarInputHandler+CallDataManager+...
4.
5.         // connection relation ((CarInputHandler,DMC), (CallDataManager,DMS))
6.         (((CCC.Port).CComponent).Port)= CarInputHandler
7.         ((CarInputHandler.CCC).Port).CComponent= DMC
8.         DMC.((CarInputHandler.CCC).Port) = CallDataManager
9.         CallDataManager.(DMC.(CarInputHandler.CCC)) = DMS
10.        ...
```

Alloy모델은 위와 같이 정의된 개념 아키텍처 모델을 검증할 수 있도록 First-Order Logic 기반의 프로퍼티(Property)를 정의할 수 있는 언어를 제공하고, 다양한 SAT 해결자(Solver)를 이용한 분석 도구를 제공한다. 따라서, 본 논문에서 정의한 일관성 규칙 (규칙 1-규칙11)을 프로퍼티로 정의함으로써, Alloy 분석 도구를 이용하여 일관성 검증을 수행할 수 있다.

```
1.  assert Rule1_2 {
2.          all c1:ConceptualArchitecture.CCP | all p1:c1.P|
3.          some c2:ConceptualArchitecture.CCP |some p2:c2.P|
4.               p2 = c2.(p1.(c1.(ConceptualArchitecture.CCC)))
5.  }
6.  check Rule1_2 for exactly 1 ELConceptualArchitecture, ...
```



(그림 3) 개념 아키텍처 모델의 검증 결과 (규칙 1.2 적용)

위의 코드 명세는 규칙 1.2를 Alloy 모델링 언어를 이용하여 정의하고 (1-5번째 줄), 이 규칙의 검증을 위해서check 명령을 이용하는 것(6 번째 줄)을 나타낸다. (그림 3)은 Alloy 검사기가 *ELConceptualArchitecture* 모델에서 *Rule1_2*의 규칙이 지켜지지 않는 부분을 찾아 반례 (counterexample)을 생성한 것을 보여준다. 이 결과가 보여주는 것은 *CarInputHandler* 컴포넌트의 *OSC* 포트는 다른 개념 컴포넌트의 포트와 어떠한 연결 관계가 없으므로 *Rule1_2*의 규칙이 지켜지지 않았다는 것을 나타낸다.

개념 아키텍처 모델 자체에 대한 검증뿐만 아니라, 개념 아키텍처 모델과 특성 모델 간의 일관성을 검증하기 위해서는 규칙 9.1를 이용하면 된다. 다음 코드 명세는 규칙 9.1를 Alloy 모델로 정의하기 위해서 두 개의 명제함수(Predicate) *DConnectedCCP*, *IConnectedCCP*와 하나의 함수 *C*를 정의하여 사용한 것을 나타낸다.

```
1.  pred DConnectedCCP[c1:CComponent, c2:CComponent] {
2.          some p1:c1.P | some p2:c2.P|
3.                  p1 = ((c1.(ConceptualArchitecture.CCC)).p2).c2
4.  }
5.  fun C[]: set (CComponent->CComponent) {
6.          {x:ConceptualArchitecture.CCP, y:ConceptualArchitecture.CCP |
7.          DConnectedCCP[x,y]}
8.  }
9.  pred IConnectedCCP[c1:CComponent, c2:CComponent] {
10.         c2 = c1.*C[]
11.}
12. assert Rule9_1 {
13.     all f1:FeatureModel.F | all f2:f1.(FeatureModel.R1)+..+f1.(FeatureModel.R8)|
14.     some c1:ConceptualArchitecture.CCP |
15.     some c2:ConceptualArchitecture.CCP |
16.         (c1!=c2) && (f1 in c1.Fcon) && (f2 in c2.Fcon) =>
17.          DConnectedCCP[c1,c2] || IConnectedCCP[c1,c2]
18.}
```

명제함수 *DConnectedCCP* 는 두 개의 개념 컴포넌트 c1과 c2 사이에 직접적인 연결관계가 성립하면 참을 반환하고, 명제함수 *IConnectedCCP* 는 c1과 c2 사이에 간접적인 연결관계가 성립하면 참을 반환한다. 함수 *C*는 직접 연결관계에 있는 모든 개념 컴포넌트들에 대한 이진 관계를 반환하는 함수로서, 명제함수 *IConnectedCCP*에서 *c1*으로부터 연속적인 이진 관계의 적용을 통해 연결된 c2가 존재하는 지를 판별하기 위해서 사용된다. 이들 함수로부터 정의된 *Rule9_1*은 *R1*에서 *R8*의 관계로 맺어진 모든 특성 *f1*과 *f2*에 대해서, 어떤 서로 다른 개념 컴포넌트 *c1*과 *c2*가 존재하여, *f1*은 *c1*에 할당

된 특성이고 *f2*는 *c2*에 할당된 특성이면 *c1*과 *c2*는 직접 혹은 간접적으로 연결되어 있어야 한다는 것을 정의한 것이다.

5.3 제품계열 프로세스 아키텍처 모델 및 검증

개념 아키텍처와 비슷하게 프로세스 아키텍처도 정의 6-8을 바탕으로 Alloy 모델로 정의하면 다음과 같다. ProcessArchitecture는 추상적인 정의이고, ELProcessArchitecture는 엘리베이터 제품계열을 위한 프로세스 아키텍처를 정의한 것의 일부분을 발췌한 것이다.

```
1. abstract sig ProcessArchitecture {
2.          PCP: set PComponent,
3.          MP2C: PComponent -> CComponent-> Feature,
4.          PCC: PComponent->PComponent
5. }
6.
7. sig ELProcessArchitecture extends ProcessArchitecture {}
8. {
9.          PCP = PCarCallHandler + PCallDataManager+...
10.
11.         // PCC = ((PCarCallHandler,PCallDataManager),...)
12.         PCarCallHandler.PCC = PCallDataManager
13.         ...
14.         // MP2C = ((PCarCallHandler, ({CarInputHandler,...}, Performance),
15.         //          (PCallDataManager, ({CallDataManager}, Performance),...
16.         (MP2C.Feature).CComponent = PCarCallHandler + PCallDataManager
17.         (PCarCallHandler.MP2C).Feature =
18.            CarInputHandler+AntiNuisanceController+WeightSensorIF
19.         CComponent.(PCarCallHandler.MP2C)=Performance
20.
21.         (PCallDataManager.MP2C).Feature = CallDataManager
22.         CComponent.(PCallDataManager.MP2C)=Performance
23.}
```

위와 같이 정의된 프로세스 아키텍처 모델에 대해서도 프로세스 아키텍처 모델 자체의 검증과 개념 아키텍처 모델 및 특성 모델과의 일관성 검증을 수행할 수 있다. 다음에 예시한 코드 명세는 프로세스 아키텍처 모델 자체의 일관성 검증을 위한 규칙 2.1과 프로세스 아키텍처와 개념 아키텍처 사이의 일관성 검증을 위한 규칙 4.2을 Alloy 모델로 표현한 것이다.

```
1. assert Rule2_1 {
2.          all p1:ProcessArchitecture.PCP | some p2:ProcessArchitecture.PCP |
3.                    p1.(ProcessArchitecture.PCC) = p2
4. }
5. assert Rule4_2 {
6.          all pc1:(ProcessArchitecture.PCC).PComponent |
7.          all pc2:pc1.(ProcessArchitecture.PCC)|
8.          some cc1:(pc1.(ProcessArchitecture.MP2C)).Feature |
9.          some cc2:(pc2.(ProcessArchitecture.MP2C)).Feature|
10.                   DConnectedCCP[cc1, cc2]
11.}
```

위의 규칙들은 5.2절에서 예시한 바와 같이 Alloy 검사기를 사용하여 모델을 검증하는 데 사용된다. 뿐만 아니라, 배치 관점과 모듈 관점의 아키텍처와 이들과 관련된 규칙도 앞에서 정의한 것과 유사하게 Alloy 모델로 정의하고, Alloy 검사기를 이용하여 일관성 검증을 수행할 수 있다.

6. 관련 연구

특성모델[3]은 영역 관점에서 소프트웨어 제품의 가변성을 분석하고 관리하기 위해서 1990년에 제안되었다. 그 이후로 여러 영역 공학 방법뿐만 아니라, 제품계열공학 방법에서도 특성모델을 제품 간의 가변성을 분석하고 관리하는 수단으로 사용하거나, 자신들의 방법에 맞게 적응시키거나 확장해왔다.

특성모델이 제품 간의 가변성을 올바르게 관리하기 위해서는 특성모델에 정의된 가변성에 대한 다양한 제약사항이 서로 상충됨이 없어야 한다. 따라서, 최근 들어 특성모델에 표현된 가변성에 대한 일관성 검사 및 다양한 추론을 제공하기 위한 연구가 많이 있어 왔다. [6]는 특성모델에 표현된 가변성에 대한 일관성을 검증하기 위해 명제 논리식을 이용하는 방법을 제안하였다. [7]에서도 [6]과 유사하게 특성모델과 명제 논리식을 대응시킴으로써 특성모델의 잘못된 가변성 모델링을 검사할 수 있는 방법을 사용하였으나, [6]과 다르게 기존 SAT 해결자(Solver)를 이용하여 자동화된 검사를 수행하는 기법을 제안하였다. [8]에서는 기존 특성모델을 비기능 특성을 포함하여 확장시키고, 이를 제약사항 해결 문제(constraint satisfaction problem)에 대응시킴으로써, 특성 모델에 대한 다양한 성질을 추론하는 방법을 제안하였다. 또한, [2]는 기존 특성 모델을 특성간의 의존성 및 특성 결합 시점 관점에서 확장한 모델을 제안하고, 이 모델의 일관성을 검사하는 규칙을 정의하였다. [2, 6, 7, 8]는 모든 특성모델에 대한 일관성 분석과 관련이 있다.

한편, 특성모델과 이를 실현하는 제품계열자산(아키텍처 혹은 컴포넌트)과의 일관성 검사를 위한 소수의 연구가 있어왔다. [9]은 주어진 특성 모델, 컴포넌트 모델, 그리고 이들 간의 대응 관계를 feature-component 모델로 정의하고, 이 모델에서 유도된 특성 모델로부터 주어진 특성 모델과의 일관성을 추론하는 기법을 제안하였다. 하지만, [9]에서는 아키텍처 모델을 컴포넌트들의 집합으로 단순화 시켰으므로, 본 논문에서 제안한 다양한 관점의 아키텍처 모델간의 일관성 검사의 적용에는 한계가 있다. 또한, [10]는 특성 모델의 특성을 구현한 컴포넌트들이 특성의 선택에 따라 안전하게 합성될 수 있는 가를 검사하는 방법을 제안하였다. 즉, 이는 특성모델과 특정 구현 모델(Feature-Oriented Programming [12])과의 일관성을 검사하는 방법으로 볼 수 있다. 본 연구에서는 특정한 구현 모델과의 일관성을 검사하는 대신에 추상화된 다양한 관점의 제품계열 아키텍처와의 일관성을 검사하기 위한 방법을 제안하였다는 점에서 차이가 있다.

7. 결 론

최근 들어 소프트웨어 제품계열의 방법 개발 및 실제 응용 소프트웨어로의 적용은 활발히 진행되어 오고 있다. 하지만, 제품계열아키텍처 개념에 대해 정형적으로 정의하고 분석하려는 노력은 거의 없어 왔다. 본 논문에서는 소프트웨어 제품계열의 핵심인 제품계열아키텍처 모델을 네 가지 관점에서 정형적으로 정의하고, 이 모델의 구성요소들이 서로 일관성을 가지는 지, 네 가지 관점의 모델 간에 일관성을 가지는 지, 특성 모델의 가변성과 일관성을 가지는 지를 검사하기 위한 규칙을 정의하였다.

제품계열아키텍처 모델이 일관성을 가진다는 의미는 이

모델로부터 개별적인 제품을 위한 유효한 아키텍처 모델을 유도할 수 있다는 의미를 가진다. 이러한 일관성을 지닌 제품계열아키텍처 모델은 특정 제품을 위한 유효한 구현 모델의 개발에 초석을 마련한다는 점에서 의의가 있다.

본 논문에서는 제품계열아키텍처의 일관성을 검사하기 위한 규칙에 대한 검증 및 타당성을 검토하기 위해서 Alloy 모델과 검사기를 사용하여 엘리베이터 제품계열의 아키텍처 모델에 대한 사례연구를 수행하였다. 이와 같은 자동화된 도구의 활용은 아키텍처 모델의 일관성 검사를 위해서 필수적인 요소이다. 하지만, 모델의 크기가 커짐에 따라서, 검증에 필요한 시간이 기하급수적으로 증가하는 문제가 발생함을 경험하였다. 따라서, 향후에는 본 논문에서 제안된 규칙을 효율적으로 검증하기 위한 방안을 개발할 것이다

참 고 문 헌

[1] P. Clements and L. Northrop, 'Software Product Lines: Practices and Patterns,' Addison-Wesley, Upper Saddle River, NJ, 2002.

[2] 이관우, "특성 지향의 제품계열분석 모델의 정형적 정의와 일관성 분석", 정보과학회논문지, 제32권 제2호, pp.119-127, 2005.

[3] K. C. Kang, S. Cohen, J. Hess, W. Nowak, S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, November, 1990.

[4] P. Kruchten, "The 4+1 View Model of Software Architecture," IEEE Software, Vol.12, Issue 6, pp.42-50, 1995.

[5] S. Thiel and A. Hein, "Systematic Integration of Variability into Product Line Architecture Design," Software Product Lines, G. J. Chastek(ed.), pp.130-153, Berlin, Springer-Verlag, 2002.

[6] M. Mannion, "Using First-Order Logic for Product Line Model Validation," LNCS Vol.2379, pp.176-187, 2002.

[7] D. Batory, "Feature models, grammars, and propositional formulas," In Proceedings of the 9th International Software Product Line Conference (SPLC), 2005.

[8] D. Benavides, P. Trinidad, A. Ruiz-Cort_es, "Automated reasoning on feature models," In Proceedings of 17th International Conference on Advanced Information Systems Engineering (CAiSE), 2005.

[9] M. Janota and G. Botterweck, "Formal Approach to Integrating Feature and Architecture Models," LNCS Vol.4961, pp. 31-45, 2008.

[10] S. Thaker, D. Batory, D. Kitchin, W. Cook, "Safe Composition of Product Lines," In Proceedings of the 6th International Conference on Generative Programming and Component Engineering (GPCE), 2007.

[11] D. Jackson, "Alloy3.0 Reference Manual," http://alloy.mit.edu/reference-manual.pdf, 2004.

[12] D. Batory, "Feature-Oriented Programming and the AHEAD tool suite," In Proceedings of the 26th International Conference on Software Engineering, pp.702-703, 2004.

이 관 우

e-mail : kwlee@hansung.ac.kr

1994년 포항공과대학교 전자계산학과(학사)

1996년 포항공과대학교 컴퓨터공학과 (공학석사)

2003년 포항공과대학교 컴퓨터공학과 (공학박사)

2003년~현 재 한성대학교 정보시스템공학과 조교수

관심분야 : 소프트웨어 제품계열 (Software Product Line), 관점지향 프로그래밍(Aspect-Oriented Programming), 소프트웨어 아키텍처