The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 14, No. 3, pp.105-110, Jun. 30, 2014. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2014.14.3.105
JIIBC 2014-3-15

휘처-휘처코드 대응을 이용한 휘처상호작용의 검출 및 모듈화

Identification and Modularization of Feature Interactions Using Feature-Feature Code Mapping

이관우*

Kwanwoo Lee*

요 약 휘처 지향 소프트웨어 프로덕트 라인 공학 방법은 휘처 단위로 프로덕트 라인의 핵심 자산을 개발하고, 이를 조합하여 쉽게 다양한 제품을 개발한다. 하지만 휘처를 조합하여 제품을 개발하는 동안 휘처상호작용문제를 효과적으로 대응하지 못하면, 개발된 제품이 원하는 대로 동작하지 않을 수 있다. 본 논문에서 휘처 간에 발생될 수 있는 원하지 않는 상호작용을 검출하는 기법과 이를 효과적으로 모듈화하는 방법을 제안한다. 제안된 방법의 적용가능성을 평가하기 위해서 공학용 계산기 프로덕트 라인에 적용하였다.

Abstract Feature-oriented software product line engineering is to develop various products by developing product line core assets in terms of features and composing those features. However, the developed product may not behave correctly if the feature interaction problem has not be properly taken into account during the feature composition. This paper proposes techniques for identifying and modularizing undesirable feature interactions effectively. The scientific calculator product line is used for evaluating the applicability of the proposed method.

Key Words : feature interaction, software product line, modularization

Ⅰ. 서 론

소프트웨어 프로덕트 라인 공학[1]은 소프트웨어 재사용 패러다임으로, 유사한 소프트웨어 제품을 개발 시에 재사용가능한 핵심자산을 개발 한 후에 이를 이용하여 개별 제품을 생산해내는 접근 방법을 취한다. 휘처 기반 소프트웨어 프로덕트 라인 공학 방법[2]은 재사용 가능한 핵심자산을 개발하기 위해서 유사한 소프트웨어 제품들 간의 공통성과 가변성을 휘처 관점에서 분석하고, 휘처 단위로 핵심자산을 모듈화 시킨 후에, 휘처의 선택에 따라 해당 휘처를 구현한 휘처 모듈을 결합하여 다양한 제품을 구성시킬 수 있다.

하지만 휘처 모듈을 결합하여 제품을 구성시킬 때, 휘처상호작용으로 인해 최종 소프트웨어 제품이 원하지 않는 행위를 나타낼 수 있다. 예를 들면, 홈 오토메이션 시스템의 경우 화재 제어 휘처는 화재 발생 시에 스프링클러를 작동시켜 화재를 진압하고, 홍수 제어 휘처는 홍수 발생 시에 메인급수를 차단하고 펌프를 작동시켜 집안에

*정회원, 한성대학교 정보시스템공학과
접수일자 2014년 5월 16일, 수정완료 2014년 6월 2일
게재확정일자 2014년 6월 13일

Received: 16 May, 2014 / Revised: 2 June, 2014
Accepted: 13 June, 2014
*Corresponding Author: kwlee@hansung.ac.kr
Dept. of Information Systems Engineering, Hansung University, Korea

들어온 물을 배수시킨다. 만약 화제와 홍수가 동시에 발생되어 두 휘처가 동시에 활성화 된다면, 홍수 제어 휘처에 의해 메인급수가 차단되어 화제 제어 휘처에서 작동시키는 스프링클러가 무용지물이 될 수 있다. 이와 같이 두 휘처 간에 발생되는 원하지 않는 행위를 휘처상호작용이라 한다. 휘처상호작용 문제는 휘처 기반으로 제품을 구성하는 휘처 지향 소프트웨어 프로덕트 라인 공학 방법에서 해결해야 할 중요한 문제이다.

지금까지 소프트웨어 프로덕트 라인 공학 방법들[2,3,4]은 휘처 기반으로 핵심자산을 모듈화 하는 기법을 많이 개발하였으나, 휘처 모듈 간의 원하지 않는 상호작용이 일어나지 않도록 혹은 원하지 않는 상호작용이 일어난 경우 이를 쉽게 해결할 수 있도록 모듈화 하는 기법은 아직까지 개발하고 있지 않다.

본 논문에서는 휘처 간의 상호작용을 검출하는 기법과 휘처상호작용을 해결할 때, 여러 모듈의 변경이 없이 최소한의 모듈 변경으로 휘처상호작용문제를 해결할 수 있는 모듈화 기법을 제안한다. 제안된 모델과 방법을 평가하기 위해 공학용 계산기 프로덕트 라인 사례를 바탕으로 실험하고 평가한다.

II. 휘처상호작용

휘처는 소프트웨어 제품 간의 구별되는 혹은 공통의 특징으로, 기능적인 서비스 특성에서부터 품질 속성, 데이터 엔터티, 제약사항 등 다양한 형태로 정의될 수 있다. 따라서 휘처 간의 상호작용은 휘처의 종류에 따라 다양한 상호작용 형태가 존재가능하나, 본 논문에서는 기능적인 특징을 나타내는 서비스 휘처의 상호작용에 국한하며, 다른 종류의 휘처 간의 상호작용[5]은 본 논문의 범위에서 벗어나며, 이는 향후 연구 범위에 포함된다.

1. 휘처상호작용의 종류

기능 휘처 간의 예상치 않는 상호작용의 종류는 크게 세 가지로 분류된다.

- 교착상태: 어떠한 휘처들도 정상적인 수행을 시작하지 못하고 대기하는 경우이다. 예를 들면, 하나의 휘처가 다른 휘처의 수행결과인 데이터를 필요로 하고, 반대로 후자도 전자의 수행결과인 데이터를 필요로 한다면 두 휘처는 서로 필요로 하는 데이터가 준비될 때까지 교착상태에 빠져서 정상적인 수행이 불가능하다.
- 불능화: 하나의 휘처가 다른 휘처의 정상적인 수행을 불가능하게 만드는 경우이다. 예를 들면, 전화 교환기 시스템의 경우 *통화중대기* 휘처와 *삼자통화* 휘처는 모두 정상적인 기능 수행을 위해서는 *삼자 연결자* 자원을 필요로 한다. 만약 *통화중대기* 휘처가 활성화되어 공유자원인 *삼자 연결자*를 사용하는 경우에 *삼자통화* 휘처는 정상적인 기능 수행을 할 수가 없다.
- 불만족: 휘처의 행위가 다른 휘처의 방해에 의해서 만족되지 않는 경우이다. 예를 들면, 엘리베이터 제어 시스템에서 *등록층정지* 휘처는 엘리베이터 운행 중에 요청이 등록된 층에서 정지하는 기능인데, 만약 *요청취소* 휘처가 엘리베이터 운행 중에 모든 엘리베이터 요청을 취소시키면 *등록층정지* 휘처는 정지할 층을 결정 못하여 그 기능을 정상적으로 종료하지 못할 수 있다.

2. 휘처상호작용의 검출

기능 휘처 간의 예기치 않는 상호작용은 휘처들이 상호 독립적이지 않고, 다양한 종류의 의존 관계가 존재하기 때문에 발생한다. 기능 휘처 간의 의존관계는 사용(Usage), 변경(Modification), 배타 활성화 (Activiation) 관계로 나뉜다. 본 논문에서는 휘처 간의 의존관계를 각 휘처에 대응된 코드 간의 의존 관계를 통해 정의한다.

정의 1. 휘처-휘처코드 대응

F는 소프트웨어 프로덕트 라인의 휘처의 집합이고, FC는 소프트웨어 프로덕트 라인의 코드 세그먼트들의 집합이라고 하자. 이때 코드 세그먼트들은 프로그램들의 추상 구문 트리 (AST:Abstract Syntax Tree)을 구성하는 클래스, 메소드, 문장, 혹은 변수들 등을 나타낸다. 휘처-휘처코드 대응 M은$F \times 2^{FC}$의 이진관계로서, $(f, CS) \in M$에서 $CS(\subseteq FC)$는 휘처 f를 구현한 프로그램 코드 세그먼트들의 집합을 나타내고, 다음의 성질을 만족해야 한다.

1. $\forall f \in F \bullet \bigcup_{(f, CS) \in M} CS = FC$
2. $\forall f \in F, \exists cs \subseteq FC \bullet (f, cs) \in M$

다음은 휘처 간의 의존관계를 정의할 때, 사용되는 술어 함수이다.

- $CS(f)$: 휘처 f를 구현한 코드 세그먼트들의 집합

- $refer(c_1,c_2)$: 코드 세그먼트 c_1에서 코드 세그먼트 c_2의 값을 참조하면 참을 반환
- $change(c_1,c_2)$: 코드 세그먼트 c_1에서 코드 세그먼트 c_2의 값을 수정하면 참을 반환
- $cflow(c)$: 코드 세그먼트 c로부터 시작되는 프로그램 제어흐름 내에 존재하는 코드 세그먼트들의 집합

정의 2. 사용 의존 (D_u)

D_u는 $F\times F$에 대한 이진관계로서, 두 휘처 f_1과 f_2는 다음의 조건을 만족할 경우에 f_1은 f_2로의 사용 의존 관계를 가진다고 정의한다.

- $\exists\, c_1\in CS(f_1), c_2\in CS(f_2)\ \bullet\ refer(c_1,c_2)$

정의 3. 변경 의존 (D_m)

D_m는 $F\times F$에 대한 이진관계로서, 두 휘처 f_1과 f_2는 다음의 조건을 만족할 경우에 f_1은 f_2로의 변경 의존 관계를 가진다고 정의한다.

- $\exists\, c_1\in CS(f_1), c_2\in CS(f_2)\ \bullet\ change(c_2,c_1)$

정의 4. 배타 활성화 의존 (D_{ex})

D_{ex}는 $F\times F$에 대한 이진관계로서, 두 휘처 f_1과 f_2는 다음의 조건을 만족할 경우에 f_1은 f_2로의 배타 활성화 의존 관계를 가진다고 정의한다.

- $\forall c_1\in CS(f_1), c_2\in CS(f_2)\ \bullet\ cflow(c_1)\cap cflow(c_2)\neq\varnothing$

이상의 휘처 의존 관계의 정의를 바탕으로 휘처 간의 예기치 않는 상호작용을 검출하기 위한 규칙을 정의할 때 사용되는 술어 함수는 다음과 같다.

- $DDep_u(f)=\{x|(x,f)\in D_u\}$: 휘처 f로부터 직접적인 사용 의존 관계에 있는 휘처들의 집합
- $DDep_u^n(f)=DDep_u\circ\cdots\circ DDep_u(f)$: 휘처 f로부터 길이 n의 패스로 간접적인 사용 의존 관계에 있는 휘처 들의 집합
- $Dep_u(f)=\bigcup_{n\in N} DDep_u^n(f)$: 휘처 f로부터 직·간접적인 사용 의존 관계에 있는 휘처들의 집합

규칙 1. 교착상태 상호작용 검출

임의의 두 휘처 f_1과 f_2는 직·간접으로 상호 사용 의존 관계에 있을 경우에 f_1과 f_2는 교착상태 상호작용을 나타낸다. 이러한 상호작용 조건을 정의하면 아래와 같다.

- $f_2\in Dep_u(f_1)\wedge f_1\in Dep_u(f_2)$

규칙 2. 불능화 상호작용 검출

임의의 두 휘처 f_1과 f_2 각각이 직·간접으로 상호 사용 의존 관계에 있는 휘처들이 있을 때, 이들 사이에 배타 활성화 의존 관계가 성립되면 f_1과 f_2는 불능화 상호작용을 나타낸다. 이러한 상호작용 조건을 정의하면 아래와 같다.

- $\exists\, x_1,x_2\ \bullet\ x_1\in Dep_u(f_1)\wedge x_2\in Dep_u(f_2)\wedge (x_1,x_2)\in D_{ex}$

규칙 3. 불만족 상호작용 검출

임의의 휘처 f_1과 직·간접으로 상호 사용 의존 관계에 있는 휘처들이 f_2로의 변경 의존 관계가 성립되면 f_1과 f_2는 불만족 상호작용을 나타낸다. 이러한 상호작용 조건을 정의하면 아래와 같다.

- $\exists\, x\ \bullet\ x\in Dep_u(f_1)\wedge(x,f_2)\in D_m$

Ⅲ. 휘처상호작용 모듈화

본 장에서는 검출된 휘처상호작용을 효과적으로 해결 및 관리하기 위한 두 가지 모듈화 패턴을 제안한다.

1. 교착상태 상호작용 모듈화 패턴

기능 휘처 간의 교착상태 상호작용은 휘처코드들이 다른 휘처코드들 내의 자원(예, 데이터)을 순환적으로 요구하기 때문에 발생된다. 따라서 의존관계에 있는 자원들을 휘처코드에서 분리하여 독립적인 모듈로 관리하는 자원 관리자를 정의한다.

교착상태 상호작용이 검출된 휘처들은 휘처를 구현하는 휘처모듈로부터 다른 휘처모듈에서 사용되는 데이터와 같은 자원들을 분리하여 *ResourceManager*에 등록시킨다. 그림 1에서 보는 바와 같이, 두개의 휘처모듈 *f1*과 *f2*는 자신들이 관리하는 자원을 *ResourceManager*에 등

록하기 위해 *addResource*() 메소드를 이용한다. 이 후에 각 휘처모듈이 *ResourceManager*에 등록된 자원을 사용하기 위해서는 getResource() 메소드나 updateResource() 메소드를 이용한다. 자원 사용을 위해 사용되는 getResource() 메소드나 updateResource() 메소드는 실질적으로 자원 사용을 수행하기 전에 자원이 사용가능한 상태인지를 검사하는 *checkAvailability*() 메소드를 내부적으로 호출한다. *checkAvailability*() 메소드는 자원 사용을 요청하는 휘처모듈들 간에 원하지 않는 상호작용이 발생되지 않도록 조정하는 정책을 구현한 것이다. 휘처모듈의 특성에 따라 상호작용을 제어하는 방법은 다양하게 구현될 수 있으므로, 이 메소드의 구체적인 구현은 다양하게 나타날 수 있다. 하지만, *ResourceManager*에 등록된 자원을 직접 사용하기 전에 이의 가용성을 검사하는 패턴은 교착상태 상호작용과 같은 휘처상호작용을 효율적으로 관리할 수 있는 중요한 설계 지침이다.

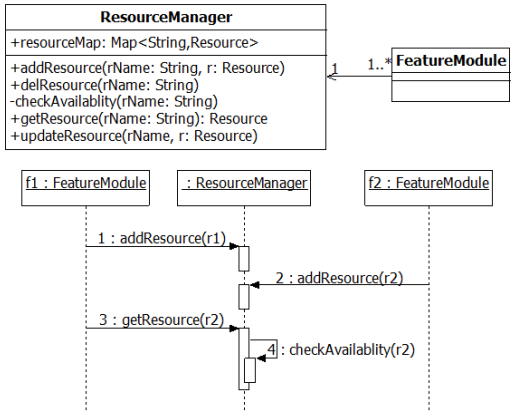

그림 1. 자원 관리자 패턴
Fig. 1. Resource Manager Pattern

2. 불능화 상호작용 모듈화 패턴

기능 휘처 간의 불능화 상호작용은 휘처모듈들이 다른 휘처모듈의 수행상태를 고려하지 않고, 자신들이 항상 수행할 수 있다는 가정 하에 동작되기 때문에 발생된다. 따라서 각 휘처모듈의 활성화 의사결정을 분리하여 독립적인 모듈로 관리하는 활성화 관리자를 정의한다.

그림 2에서 보는 바와 같이, 각 휘처모듈은 자신의 기능을 수행하기에 앞서, *checkPermission()* 메소드를 통해서 자신의 수행여부를 *ActivationManager*에 문의한다. 만약 수행여부가 허가된다면 휘처모듈은 수행중에 자신의 수행상태의 변경을 *changeState*() 메소드를 통해서 *ActivationManager*에 지속적으로 알려주고, 이때, *ActivationManager*는 *notify()* 메소드를 호출하여 이전에 수행여부를 문의한 다른 휘처모듈 중에 수행 가능한 휘처모듈이 있는 지를 판정하여, 해당 휘처모듈이 있는 경우 이를 수행시킨다. *checkPermission()*과 *notify()* 메소드는 상호작용을 하는 휘처모듈 간의 활성화 의존성을 바탕으로 구현되어야 한다. 따라서 이 메소드들의 구현은 상호작용에 참여하는 휘처모듈에 따라 달리 구현될 수 있다.

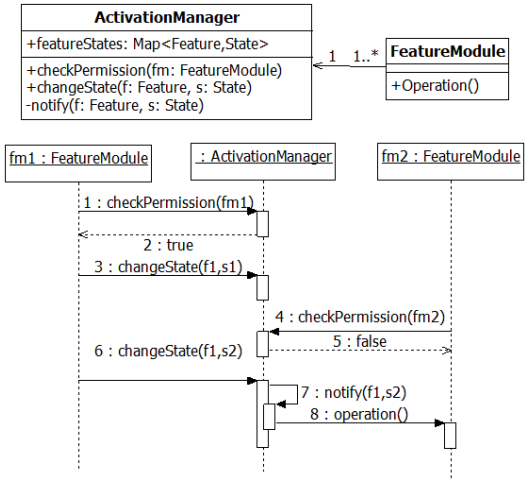

그림 2. 활성화 관리자 패턴
Fig. 2. Activation Manager Pattern

3. 불만족 상호작용 모듈화 패턴

기능 휘처 간의 불만족 상호작용은 하나의 휘처모듈들이 다른 휘처모듈에 의해서 변경될 수 있는 상태를 내부에 포함하고 있고, 이들 상태는 오로지 자신의 휘처모듈에 의해서만 변경될 수 있다는 가정 하에 구현되어 있기 때문에 발생된다. 따라서 다른 휘처모듈의 상태를 변경시키는 휘처모듈은 변경요청을 직접 해당 모듈에 보내기 보다는 변경 요청을 변경 프록시에 보내에 변경 프록시가 변경 요청을 해당 모듈에 전송할지 말지를 결정하도록 함으로써 원하지 않는 상호작용 문제를 해결할 수 있다.

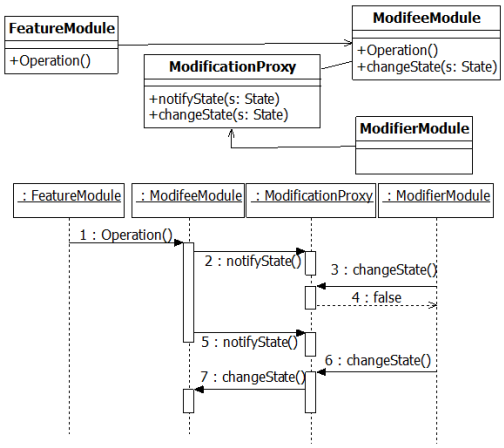

그림 3. 변경 프록 패턴
Fig. 3. ModificationProxy Pattern

그림 3에서 *ModifierModule*은 다른 휘처모듈의 상태를 변경시키는 휘처모듈을 나타내고, *ModifeeModule*은 다른 휘처모듈에 의해서 자신의 상태 및 행위가 변경되는 휘처모듈을 나타낸다. *ModifeeModule*이 다른 휘처모듈에 의해서 사용되고 있는 동안에 *ModifeeModule*은 자신의 상태변화를 *notifyState*() 메소드를 통해서 *ModificationProxy*에 공지한다. 만약 *ModifeeModule*이 다른 휘처모듈에 의해서 사용되는 중에 *ModifierModule*이 *ModifeeModule*의 상태를 변경시키고자 한다면, 직접 변경을 요청하지 않고, *ModificationProxy*를 통해서 변경을 요청하고 *ModificationProxy*는 *ModifeeModule*의 상태를 고려하여 변경요청을 *ModifeeModule*에 전달할지 말지를 결정한다.

Ⅳ. 사례 연구 및 평가

본 장에서는 앞에서 설명한 휘처상호작용 검출 및 모듈화 기법을 평가하기 위해서 공학용 계산기 프로덕트 라인에 적용한 사례를 설명하고, 적용사례 결과로부터 제안된 방법을 평가하고자 한다.

공학용 계산기 프로덕트 라인[6]은 자바 언어로 구현된 오프소스 프로그램인 Java Scientific Calculator를 휘처 단위로 모듈화한 것이다. 공학용 계산기 프로덕트 라인은 12개의 휘처모듈로 분해되어 있다.

공학용 계산기 프로덕트 라인의 휘처상호작용을 검출하기 위해서 휘처상호작용 검출 시스템 프로토타입을 개발하였다. 개발된 프로토타입은 오픈 소스인 Java System Dependency Graph API를 이용하여 2장에서 정의된 휘처모듈 간의 의존성과 휘처모듈 간의 상호작용 검출 모듈을 개발하였다.

표 1. 휘처 의존성
Table 1. Feature Dependency

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2										M U		
3											U	
4								E				
5											U	
6	E							E M	M	M		
7						M		E		M	M U	
8	M E										M	
9								E		M	M	
10	M U										M	
11	M							E	M	M		
12												

1.common, 2.booleanOps, 3.combinatoricOps, 4.editing
5.exponetionalOps, 6.history, 7.memory, 8.mode, 9.notation,
10.numberSystems, 11.shiftKey,12.trigonometircOps
D: 사용 의존성, M: 변경 의존성, E: 배타활성화 의존성

표 2. 휘처상호작용
Table 2. Feature Interaction

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2			UI			UI	UI	**UI**	UI		**UI**	
3							UI	UI	UI	UI		
4												
5							UI	UI	UI	UI		
6										**DI**		
7								UI	UI	UI		
8										UI **DI**		
9												
10											UI	
11												
12												

DI: 불능화 상호작용, UI: 불만족 상호작용

표 1은 개발된 프로토타입을 이용하여 12개 휘처모듈 간의 의존성을 분석한 결과이다. 표 1의 각 셀에 표시된 문자는 각각 사용(U), 변경(M), 배타활성화(E)를 나타낸

다. 또한, 표 2는 표 1의 의존성 정보를 바탕으로 휘처상호작용을 검출한 결과이다. 표 2의 각 셀에 표시된 문자는 각각 불만족(UI), 불능화(DI) 상호작용을 나타낸다. 표 2에서 보는 바와 같이 공학용 계산기 프로덕트 라인에서는 교착상태 상호작용은 검출되지 않았고, 19개의 불만족 상호작용과 2개의 불능화 상호작용이 검출되었다.

이를 바탕으로 21개의 휘처상호작용을 효과적으로 모듈화하기 위해서 3장에서 제안된 패턴을 적용하여 공학용 계산기 프로덕트 라인을 리팩토링하였다. 리팩토링 전의 코드 사이즈는 6.5 KLOC 이었고, 리팩토링 후, 휘처상호작용을 관리하기 위해 추가된 21개의 모듈로 인해 코드 사이즈가 6.8 KLOC로 약 4.6% 증가하였지만, 휘처상호작용을 관리하는 코드를 휘처모듈에서 분리하여 설계하였기 때문에, 새로운 휘처상호작용관리 정책이 고려되는 경우에 기존 휘처모듈의 코드를 변경하지 않고, 효과적으로 이를 반영시킬 수 있는 장점이 있다.

V. 결 론

본 논문에서는 휘처 단위로 프로덕트 라인 자산을 개발할 때 발생될 수 있는 휘처상호작용문제를 체계적으로 해결할 수 있는 방법을 개발하였다. 먼저, 휘처 간의 의존성정보를 휘처를 구현한 모듈 간의 의존관계를 통해 추출하고, 추출된 휘처 간의 의존성 정보를 바탕으로 휘처 간의 예기치 않게 발생될 수 있는 상호작용을 추출하였다. 또한 휘처상호작용을 처리하는 코드를 효과적으로 모듈화하는 방안을 제안함으로써, 다양한 휘처상호작용 해결 정책을 적용하는 경우에도 일부의 모듈 변경으로 쉽게 새로운 휘처상호작용 해결 정책을 적용할 수 있게 하였다.

제안된 기법의 타당성을 평가하기 위해 기존에 휘처 단위로 모듈화된 공학용 계산기 프로덕트 라인에 적용하여 적용가능성을 실험하였지만, 논문에서 제안된 기법의 일반화를 위해서는 다양한 분야의 프로덕트 라인 시스템에 적용할 예정이다.

References

[1] P. Clements, L. Northrop, “Software Product Lines: Practices and Patterns”, Addison-Wesley, 2002

[2] S. Apel, D. Batory, C. Kastner, G. Saake, “Feature-Oriented Software Product Lines”, Springer, 2013

[3] K. Kang, J. Lee,, P. Donohoe, “Feature-Oriented Product Line Engineering”, Software, vol. 19, no. 4, pp. 58-65, July/August 2002.

[4] H. Gomaa, “Designing Software Product Lines With UML: From Use Cases to Pattern-Based Software Architectures”, Addison-Wesley, 2005

[5] K. Lee, “A Method for Deriving an Optimal Product Feature Configuration Considering Feature Interaction”, The Journal of IIBC. Vol.14 No.2, pp.115-120, Apr. 30, 2014

[6] K. Lee, G. Botterweck, S. Thiel. "Aspectual separation of feature dependencies for flexible feature composition." Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International. Vol. 1. IEEE, 2009.

[7] http://jscicalc.sourceforge.net.

[8] http://www4.comp.polyu.edu.hk/~cscllo/teaching/SDGAPI/.

저자 소개

이 관 우(정회원)

- 2003년 : POSTECH, 박사
- 2008년 : University of Limerick, 방문교수
- 2003~현재 : 한성대학교 정보시스템공학과, 부교수
- 주관심분야 : Software Product Line Engineering, M2M, Aspect-Oriented Programming, Software Architecture

※ 본 연구는 한성대학교 교내연구비를 지원 과제임.