컴포넌트 가변성 유형 및 Scope에 대한 정형적 모델

(A Formal Model of Component Variability Types and Scope)

소 동 섭 † 신 규 석 †† 김 수 동 †††
(Dong-Seob So) (Kyoo-Seok Shin) (Soo-Dong Kim)

요 약 시스템 개발 시 복잡성을 감소시키고 개발 비용과 기간을 단축하기 위하여 컴포넌트 기반 개발(CBD)이 산업계에서 보편화 되고 있다. 그러나, 현재 대표적인 CBD는 한 회사나 하나의 어플리케이션에 의존적인 컴포넌트를 개발하고 있다[1,2]. 따라서 어떤 도메인의 여러 패밀리에서 컴포넌트를 재사용하기 위한 컴포넌트 가변성이 강조 되고 있다. 하지만, 컴포넌트 가변성의 정의나 유형에 대해 구체적으로 제시된 연구가 미흡하여, 컴포넌트의 주 목적인 재사용 측면보다, 유지보수 목적의 컴포넌트가 개발되고 있다[3]. 본 논문에서는 컴포넌트의 재사용성을 높이기 위하여, 컴포넌트 특징을 반영한 컴포넌트 가변성을 정형적으로 정의한다. 또한, 기존의 컴포넌트 가변성으로 인식된 논리 가변성을 명확하게 정의하고, 추가로 3가지 가변성 유형을 제시함으로써, 컴포넌트에서 존재하는 모든 가변성 유형을 제시한다. 컴포넌트 커스터마이제이션시에 가변성의 경우의 수를 예측, 확인 할 수 있는 컴포넌트 가변성 Scope를 제시한다. 컴포넌트 개발에 있어서 이러한 기법을 적용함으로써, 여러 패밀리 멤버에서 재사용 할 수 있는 고품질의 컴포넌트 개발을 지원 할 수 있다.

키워드 : CBD, 컴포넌트, 가변성 유형, 가변성 Scope, 정형명세

Abstract Component-based development(CBD) has been generalized in industry to master the complexity and reduce the development cost and time. However, current CBD practice is developing the component which is dependent on single application[1][2]. Therefore component variability is emphasized to reuse the component in many family members in a domain[8]. However, components are developed for the reason of replaceability rather than the reusability which is the main purpose of the component due to the insufficiency of the study of component variability definition and type[3]. In this paper, we formally specify the component variability reflecting the characteristics of the component to increase the component reusability. We define the logic variability which was recognized as the existing component variability and we propose all types of variability existing in the component by suggesting three more variability types. And we propose the component variability scope which makes us estimate and verify the number of cases of the variability when we customize the component. We propose these component variability types and scope through formal specification. By applying these techniques in developing components, we can develop high quality components reusable in many family members.

Key words : CBD, Component, Variability Types, Variability Scope, Formal Specification

1. 서 론

객체단위의 소프트웨어 재사용의 한계를 극복하고자 컴포넌트 단위의 재사용이 요구 되었다. 컴포넌트 단위의 재사용은 시스템 개발 시 복잡성을 감소시키고 재사용성을 증가시켜, 시스템 개발 비용과 기간을 단축 할 수 있는 효율적인 방법이다[4]. 따라서 컴포넌트의 재사용성을 높이기 위한 많은 연구가 수행 되고 있다. 그 중

· 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 비 회 원 : 숭실대학교 대학원 컴퓨터학과
dsso@otlab.soongsil.ac.kr
†† 비 회 원 : 한국정보통신기술협회
skshin@tta.or.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2002년 3월 18일
심사완료 : 2002년 10월 1일

에서 공통성과 가변성 분석은 컴포넌트의 재사용성을 높이기 위한 기법으로 평가 되고 있다.

그러나 기존의 CBD에서는 컴포넌트 가변성에 대한 연구가 미흡하여, 어떤 도메인의 여러 패밀리에서 재사용 되는 컴포넌트 보다는 한 회사나 하나의 어플리케이션에 의존적이며, 유지보수 목적의 컴포넌트가 개발되어 왔다[5,6]. 또한, 컴포넌트 가변성의 정의를 기존의 소프트웨어 공학에서의 일반적인 가변성의 정의 및 프로그래밍 언어에서의 가변성 또는 객체지향 모델링에서의 가변성의 정의를 사용해 왔다.

컴포넌트에 존재하는 여러 유형의 모든 가변성을 식별하기 위해서는 기존의 개략적으로 알려진 다른 알고리즘이나 로직을 사용하는 논리 가변성 이외에, 추가적인 가변성 유형이 제시되어야 한다. 또한, 컴포넌트를 각각의 패밀리에서 재사용 하기 위하여 커스터마이제이션을 할 경우 컴포넌트 가변성이 일어 날 수 있는 경우의 수를 표현한 컴포넌트 가변성 Scope 정의가 필요하다.

따라서 본 논문에서는 어떤 도메인의 여러 패밀리에서 사용되는 재사용성이 높은 컴포넌트를 개발하기 위하여, 컴포넌트 특징을 반영한 컴포넌트 가변성을 정형적 명세를 통해 정의한다. 기존의 개략적으로 알고 있는 논리 가변성 유형을 명확하게 정의하고, 추가로 속성 가변성과, 워크플로우 가변성 그리고, 데이타 영구 가변성을 제안한다. 또한, 컴포넌트 가변성 Scope를 이진 Scope와 선택 Scope 그리고 Unknown Scope로 분류한다.

본 논문의 구성은 제2장에서 기존의 기법에서 제안한 가변성을 소개하고 각각의 가변성이 컴포넌트에 적용될 때 갖는 한계점을 살펴본다. 제3장에서는 정형명세를 통해 컴포넌트 가변성을 정의 한다. 제4장에서는 컴포넌트 가변성의 유형 분류를 제시하고 제5장에서는 컴포넌트 가변성 Scope에 대해 제시한다. 제6장에서는 논문에서 제시한 컴포넌트 가변성 유형 및 Scope를 평가한다. 제7장에서는 결론을 제시한다.

2. 관련 연구

2.1 KobrA Method[7]

KobrA 방법론은 독일의 Fraunhofer Institute for Experimental Software Engineering(IESE)에서 개발된 컴포넌트 개발 방법론이다. KobrA 방법론은 소프트웨어 재사용 문제의 해결책으로써 제시된 주요 기술인 컴포넌트 기반 개발방법론(CBD)과 아키텍쳐 스타일 및 디자인 패턴과 Product Line 공학의 기술들을 포함한 방법론이다.

KobrA 방법론에서는 어플리케이션들 사이에서 발생하는 차이점들을 가변성 이라 정의한다. 이 방법론에서는 가변성을 두 유형으로 분류하고 있는데, 실행 시간 가변성과 개발 시간 가변성이다. 실행시간 가변성이란, 소프트웨어의 실행 시에 입력되는 값에 따라 실행 경로가 가변적인 것을 말한다. 예를 들면, 도서관에 비치되어 있는 책과 사용자에게 대출 되지 않은 책은 실행 환경 가변성이다. 이러한 실행 환경 가변성은 소프트웨어의 본질적인 부분이며, 전통적인 설계와 프로그래밍 기법으로 처리 될 수 있다.

KobrA 방법론에서는 두 가지의 프로세스를 가지고 있는데 프레임워크 프로세스와 어플리케이션 프로세스이다. 프레임워크 공학에서는 여러 패밀리 멤버들과 유사한 어플리케이션을 지원하기 위한 가변성과 가변성을 결정짓는 결정 모델(Decision Model)의 하부 액티비티가 정의 되어 있다.

KobrA 방법론은 우선 가변성의 두 가지 유형인 실행 시간 가변성과 개발 시간 가변성을 구분 할 수 있는 기준 개념이 중복되는 부분이 있다. 실행 시간 가변성은 소프트웨어를 개발하는 동안에 추출되는 것으로써, 개발 시간 가변성을 통해서만 실행 시간 가변성을 추출 할 수 있다. 따라서 가변성의 유형 분류에 대한 정의가 불분명 하다. 둘째, 가변성을 추출하는 액티비티와 그 절차가 명확하게 정의되어 있으나 상세한 세부 지침들이 요구된다. 셋째, 가변성을 표현 하는 방법이 각각의 모델에서 <<variant>> 라는 하나의 스테레오 타입으로 보여지고 있다. 가변성 유형에 해당하는 표현 방법이 필요하다.

2.2 Family-Oriented Abstraction, Specification and Translation (FAST)[8]

FAST는 Lucent에서 개발한 Product-Line 공학 방법론이다. FAST의 핵심은 시스템의 패밀리에서 공통성을 분석하는 액티비티 단계이다. 공통성 가운데 가변성들이 비슷한 문장 형태로 찾아지고, 패밀리에서 공통성과 가변성을 특징짓는 문서를 작성한다. FAST에서는 패밀리 멤버들이 다른 멤버와 어떻게 다른지에 대한 가정을 가변성이라 한다. 가변성 매개변수 테이블을 통해 가변성 매개변수와 매개변수의 의미, 값의 범위, 바인딩 시간, 기본값을 표현 한다.

공통성 분석을 통해 시스템 개발 환경이 만들어 진다. 시스템 개발 환경이 만들어지면, 새로운 소프트웨어의 개발 비용을 줄이면서 빠른 시간에 개발할 수 있다. FAST는 패밀리, 공통성, 가변성에 대해 정의하고 있으며, 공통성과 가변성을 찾기 위한 도메인의 공통성과 가

변성 설립이라는 액티비티를 정의하고 있다. 또한 각 액티비티의 일관성을 유지하기 위한 분석, 평가 방법을 제시하고 있다. FAST는 공통성과 가변성을 상세하게 기술하는 양식이 주어져 있다.

그러나 FAST 는 Product Line 공학에서의 문제점들을 상위 수준의 추상화를 통해 다루고 있다. 또한 FAST에서는 가변성들 사이의 분류 및 상세화 그리고 가변성들 사이의 카테고리의 정의가 충분치 않아 가변성들 사이의 관계와 유형에 대한 추가적인 정의가 필요하다. 따라서 컴포넌트에 FAST에서 정의한 공통성과 가변성을 적용하기 위해서는 컴포넌트의 특징에 맞도록 그 정의를 보완할 필요가 있다.

2.3 Component Modeling Technique (COMO)[9]

COMO 방법론에서는 실용적인 객체지향의 컴포넌트 개발 방법론을 제안 한다. COMO는 컴포넌트 개발을 위해 Unified Modeling Language(UML) 과 Rational Unified Process를 확장하였다. 따라서 COMO는 체계적인 절차와 각각의 개발 태스크를 포함한 지침을 제공한다. COMO에서는 컴포넌트를 개발하기 위하여 도메인 분석 단계와 컴포넌트 디자인 단계로 나누고 각각의 단계에서 수행해야 할 태스크들을 정의하는데 중점을 두고 있다.

컴포넌트 모델링의 업무 흐름을 크게 도메인 요구사항 집합 식별, 공통성 식별, 가변성 식별, 개략 객체 모델 생성, 컴포넌트 식별, 컴포넌트에 비즈니스 객체 할당, 컴포넌트 요구사항 명세서 정의, 모델정제로 분류하고 있다.

COMO 방법론의 가변성 식별 단계에서는 소프트웨어 공학에서 정의한 가변성 정의인 "어플리케이션에서 변경 될 수 있는 부분"을 인용하여 사용하고 있다. 따라서 컴포넌트의 특징을 반영한 컴포넌트 가변성의 한계가 있다. 또한 가변성의 유형 분류에 대한 기준이 제시되어 있지 않다. COMO 방법론은 컴포넌트를 개발하기 위한 프로세스에 중점을 두고 기술한 것이기 때문에, 컴포넌트 가변성 유형에 대한 제안만 있을 각 유형에 대한 정확한 설명이나 의도하는 바가 기술 되지 않았다.

그러나 컴포넌트 모델링 프로세스를 제시하는데 중점을 두고 있어, 컴포넌트 가변성 유형들에 대한 구체적인 정의와 유형간에 구분 할 수 있는 기준이 충분히 기술되어 있지 않아, 컴포넌트 가변성 추출 지침을 유도하기에 어려움이 있다.

2.4 Bachmann 기법[10]

Bachmann 기법은 오랜 기간 동안 소프트웨어 제품에서 변화 되어 왔던 부분을 찾아내어 그 부분을 표현하고, 아키텍쳐내에서 변경될 부분의 위치를 예측하여 명확하게 보여주는 것이 목적이다. Bachmann은 아키텍쳐 내에서 가변성을 관리하기 위한 방법과 고객이 제품을 주문했을 때, 고객의 서로 다른 요구사항들을 가변성 발생 시점으로 보고, 이러한 가변성이 아키텍쳐 내에서 어떻게 포함되는가에 대한 방법을 제시 한다.

Bachmann기법에서는, 아키텍쳐 설명서에서 보이는 소프트웨어 품질 관점에서 가변성이 발생할 수 있는 부분을 제시하고, 제품간의 가변성 관계를 표현하는 방법을 제시 하였다. 아키텍쳐 수준에서 가변성을 표현하는 방법에 대해 중점을 두고 있다.

소프트웨어 아키텍쳐에서 가변성이 일어나는 원인을 여섯 가지로 분리 하였다. 첫째, 기능 부분이다. 특정 기능이 몇몇 제품에서는 존재하고 다른 제품에는 존재 하지 않는 경우이다. 기능 존재의 유무가 가변적인 경우이다. 둘째, 데이타 부분이다. 특정 데이타의 구조가 제품에 따라 다른 경우 이다. 셋째, 제어 흐름 부분이다. 시스템 레벨의 제어 흐름에서 발생하는 가변적인 부분으로 제품간에 데이타의 일관성 유지나 에러 복구를 위한 패턴이 다른 경우를 의미한다. 넷째 기술 부분이다. 운영체제, 하드웨어, 미들웨어, 사용자 인터페이스, 프로그램 언어의 실행 환경과 같은 플랫폼에서 제품마다 필요한 기능이 다양한 경우이다. 다섯째는 품질 목적 부분이다. 제품마다 중요시 하는 품질 특성에 따라 사용해야 할 데이타나 기능이 다른 경우이다. 여섯째, 환경 부분이다. 제품 환경에 따라 제품과 상호 작용하는 방식이 다른 경우이다.

Bachmann에서는 가변성의 유형을 세 가지로 제시하고 있다. 첫째, 한 제품에는 기능이 존재하지만, 다른 제품에서는 존재하지 않는 경우이다. 둘째, 가변성의 선택이다. 제품의 사양에 따라 가변적인 부분을 선택 할 수 있는 유형이다. 셋째, 가변성의 선택 집합이다. 예를 들면 한 제품이 동시에 여러 프로토콜을 지원해야 하는 경우, 프로토콜의 집합을 선택하는 경우이다.

Bachmann에서 제시한 가변성은 소프트웨어 아키텍쳐를 구성하는 제품 사이에서 서로 다른 부분이거나, 변화가 일어날 부분이다. 따라서 재사용성을 향상시키기 위하여 여러 패밀리 멤버의 공통 요구 사항 중에서, 발생하는 컴포넌트 가변성과는 목적과 비교 대상이 다르다.

2.5 Feature-Oriented Domain Analysis (FODA)[11]

FODA는 Software Engineering Institute(SEI) 에서 개발한 가장 잘 알려진 Product Line 방법론이다. FODA의 주 산출물은 도메인공학을 적용한 Feature 모

델이다. FODA에서 Feature는 소프트웨어 시스템 사용자 측면에서 봤을 때 그 소프트웨어 시스템만이 갖는 특성으로 도메인의 공통성과 가변성을 모두 포함하는 개념이다. 이 Feature 라는 개념을 사용하여 시스템의 공통성과 가변성을 분석한다. FODA의 핵심적인 아이디어는 도메인을 분석하여 이 도메인 안에서 시스템이 제공하는 Feature를 식별하는 것이다.

도메인으로부터 식별된 Feature는 Feature모델 내에서 계층적으로 표현된다. Feature는 다른 Feature의 집합으로 구성된다. Feature에 따라 시스템에 포함 여부가 결정되는 선택적인 경우와 반드시 포함되어야 하는 필수 경우로 분리 된다. Feature는 어떠한 다른 Feature에 포함 되어야 한다. Feature들 사이에는 집합관계, 상속관계, 매개 변수화 관계의 세 가지 관계가 존재한다. 이러한 관계 상세화는 Feature 트리(Tree)로 표현된다. FODA는 Feature를 식별하기 위해 기능, 운영, 표현 세 개의 분류로 나누고 있다. 또한 Feature를 상세화하고 Feature의 유형을 분류하였다. Feature에 대한 분류와 형식, Feature를 작성하기 위한 다이어그램이 준비되어 있어 개발자들은 Feature를 쉽게 식별하고 상세화할 수 있다.

FODA 에서는 직관적인 방법을 사용하여 소프트웨어 시스템의 공통성과 가변성을 Feature Tree를 사용하여 추출 한다. 공통성과 가변성을 포함한 Feature 위주의 도메인 분석을 컴포넌트에 적용하기 위해서는 컴포넌트를 위한 특징의 정의가 반영되어야 한다. 또한 Feature에서 공통성과 가변성을 식별할 수 있는 판단 기준이 필요하다.

3. 컴포넌트 가변성의 정의

소프트웨어공학에서는 어플리케이션마다 변경 될 수 있는 부분을 가변성이라 정의 한다[12]. 프로그램 언어에서의 가변성은 데이타 유형으로 선언된 변수의 값들 사이의 가변성을 말한다. 즉, Account라는 String 타입의 계좌변수에 01-345-57887-09와 01-347-65872-11이라는 여러 값들이 입력 되는 것을 가변성이라 한다.

객체지향 모델링에서의 가변성은 동일한 오퍼레이션 이름을 갖지만 인자의 타입과 개수에 따라 조건에 맞는 오퍼레이션을 실행하는 오버로딩 장치의 오퍼레이션 가변성과 다형성 장치를 통한 클래스 가변성 등이 있다.

하지만, 이러한 기존의 가변성들은 소프트웨어공학에서의 일반적인 가변성이나, 프로그램 언어에 의존적인 관점에서 다루어지고 있다. 또한 객체지향 모델 기법에서는 객체지향 프로그램의 장치에서의 가변성 구현 관점에 중점을 두고 있다. 따라서, 어떤 도메인의 여러 패밀리에서 컴포넌트를 재사용할 수 있도록, 컴포넌트를 이용, 조립 또는 커스터마이제이션 하는 과정에서 요구되는 컴포넌트 가변성의 특징에 대해서는 다루어 지지 않고 있다.

본 논문에서는 이러한 컴포넌트의 특징을 컴포넌트 가변성에서 반영하고자 다음과 같이 컴포넌트 가변성을 기술한다.

도메인이란 어떤 분야의 업무 참가자들에게 이해되는 용어나 개념들의 집합에 의해 특징 되는 활동이나 지식들의 영역이다[13]. 즉, CBD에서의 도메인은 컴포넌트 모델링을 하기 위한 산업부문으로 예를 들면 은행 분야가 선택 되었다면 이 은행 분야가 도메인이 된다.

(정의 1) *Domain*이란 컴포넌트 모델링을 하기 위한 산업부문.

패밀리란 동일한 도메인에서 컴포넌트 모델링을 하기 위한 대상 회사들의 집합이다. 예를 들면 은행 도메인에는 수많은 은행들이 있지만, 그 중에서 컴포넌트 모델링의 대상이 되는 은행 가운데 국민, 한빛, 신한 은행들이 패밀리이다.

(정의 2) *Family*란 동일한 도메인에서 컴포넌트 모델링을 하기 위한 대상 회사들의 집합

따라서 은행 도메인에 패밀리인 국민, 한빛, 신한은행이 포함 된다.

$$Family \subset Domain$$

패밀리멤버란 이러한 패밀리를 구성하는 각각의 멤버이다. 예를 들면 국민은행, 한빛은행, 신한은행 각각이 패밀리멤버이다.

(정의 3) *FamilyMem*는 패밀리를 구성하는 각각의 멤버

패밀리멤버는 패밀리를 구성하는 각각의 멤버이므로 패밀리멤버는 패밀리의 원소이다.

$$FamilyMem \in Family$$

$$Family = \{FamilyMem_i \mid 1 \le i \le n \text{, n은 패밀리멤버의 개수}\}$$

각각의 패밀리멤버에는 요구사항들이 있다. 패밀리멤버 요구사항은 각각의 패밀리멤버가 갖는 요구사항이다. 예를 들면 패밀리멤버인 국민은행은 국민은행의 요구사항을 가진다.

(정의 4) $\exists\, i \in Family$, RF_i는 i가 갖는 요구사항

패밀리멤버 요구사항은 기능을 수행하는 로직이나 알고리즘과 데이타를 나타내는 속성과 작업을 수행하기 위한 기능들간의 흐름인 워크플로우, 데이타의 논리적인 집합인 데이타 스키마를 포함한다.

$RF_i \Rightarrow RFOfLogic_i \cup RFOfAttribute_i \cup$

$RFOfWorkflow_i \cup RFOfPersistent_i$

패밀리 요구사항 집합은 각각의 패밀리멤버들의 요구사항을 모은 것이다. 예를 들면, 국민은행의 요구사항과 한빛은행, 신한은행의 요구사항을 모두 모으면, 패밀리 요구사항 집합이 된다.

(정의 5) $SetOfRF = \{RF_i \mid i \in Family\}$

각각의 패밀리멤버의 요구사항들 사이에서 공통적인 부분은 재사용성이 가장 높은 부분으로써, 컴포넌트 모델링의 대상이 된다. 이 부분이 컴포넌트 공통성 부분이다.

(정의 6) *CommonSetOfRF*는 패밀리멤버의 요구사항들 중 공통적인 부분

따라서 컴포넌트 공통성은 다음과 같은 조건을 만족한다.

$\forall i \neq j \neq k, (CommonSetOfRF(i,j,k) \supset (RF_i \cap RF_j \cap RFk)) \wedge (CommonSetOfRF(i,j,k) \subset ((RF_i \cap RF_j) \cup (RF_i \cap RF_k) \cup (RF_j \cap RF_k)))$

각각의 RF_i, RF_j, RF_k는 임의의 패밀리멤버 요구사항이다.

컴포넌트 공통성은 모든 패밀리멤버 요구사항들의 교집합을 반드시 포함하고, 각각의 패밀리멤버의 요구사항들 사이에 교차하는 범위에 포함될 수도 있다. 예를 들면 그림 1에서 RF_i, RF_j, RF_k의 세 패밀리멤버 요구사항이 교차하는 부분은 재사용성이 가장 높은 부분으로 반드시 포함되며, RF_i와 RF_j 및, RF_i와 RF_k 그리고, RF_j와 RF_k가 교차하는 요구사항도 컴포넌트 공통성부분으로 포함된다.

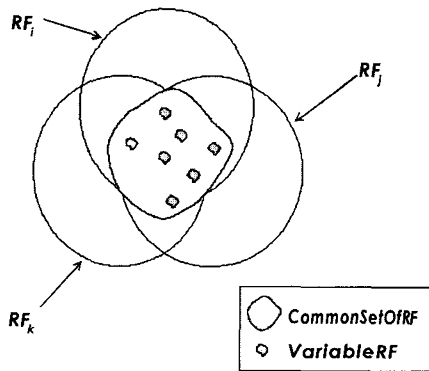

그림 1 컴포넌트 가변성 식별영역

컴포넌트 공통성은 패밀리 요구사항 집합에 포함된다.

$CommonSetOfRF \subset SetOfRF$

각각의 패밀리멤버의 요구사항들의 집합이 패밀리 요구사항 집합의 멤버이므로 패밀리 요구사항의 포함되는 컴포넌트 공통성은 패밀리멤버의 기능을 수행하는 로직이나 알고리즘과 데이타를 나타내는 속성, 작업을 수행하기 위한 기능들간의 흐름인 워크플로우, 데이타의 논리적인 집합인 데이타 스키마를 포함한다.

$RF_i \in SetOfRF$

$CommonSetOfRF \subset SetOfRF$

$\forall RF_i \in CommonSetOfRF$

따라서, *CommonSetOfRF*는 다음의 조건이 성립한다.

$CommonSetOfRF \Rightarrow CommonSetOfRFLogic \cup CommonSetOfRFAttribute \cup CommonSetOfRFWorkflow \cup CommonSetOfRFPersistent$

컴포넌트 공통성에 포함된 패밀리멤버 요구사항들은 각각의 패밀리멤버들 사이에 동일한 기능들을 가지고 있다. 이 동일한 기능을 각각의 패밀리멤버에서 수행할 때 논리적 측면, 속성 측면, 워크플로우 측면, 데이타 영구 측면에서 다른 부분이 존재한다.

(정의 7) G_i는 컴포넌트 공통성에 포함된 패밀리멤버 요구사항들의 동일한 기능, i는 어떤 패밀리멤버

G_i는 논리적인 측면($G_i.logic$)과 속성 측면($G_i.attribute$), 워크플로우 측면($G_i.workflow$), 데이타 영구 측면($G_i.persistent$)을 포함한다. 따라서 다음과 같이 표현된다.

$Gi \Rightarrow G_i.logic \cup G_i.attribute \cup G_i.workflow \cup G_i.persistent$

컴포넌트 가변성은 각각의 패밀리멤버가 Gi 를 수행하는 방법이 다른 경우 이다.

(정의 8) *VariableRF*는 컴포넌트 공통성에 포함된 패밀리멤버 요구사항들의 동일한 요구 사항들이 각각의 패밀리멤버에서 다르게 수행하는 경우.

따라서 컴포넌트 가변성의 집합은 그림 1의 *Variable RF*의 집합이며 다음과 같이 표현 된다.

(정의 9) *SetOfVariableRF*는 컴포넌트가변성의 집합

$SetOfVariableRF=\{G_i \mid G_i \in CommonSetOfRF,\ G_i.logic(FamilyMem_j) \neq G_i.logic(FamilyMem_k) \vee G_i.attribute(FamilyMem_j) \neq G_i.attribute(FamilyMem_k) \vee G_i.workflow(FamilyMem_j) \neq G_i.workflow(FamilyMem_k) \vee G_i.persistent(FamilyMem_j) \neq G_i.persistent(FamilyMem_k)\},$

$\exists i \neq j \neq k(FamilyMem_i,\ FamilyMem_j,$

$FamilyMem_k) \in Family$

4. 컴포넌트 가변성의 유형 분류

본 논문에서는 컴포넌트 가변성을 크게 네 가지 유형으로 분류한다. 이 유형 분류는 컴포넌트 개발 과정 중 분석 단계에서 컴포넌트 가변성을 추출하고, 가변성의 여부를 판단하는 기준으로 사용 된다. 컴포넌트 가변성을 네 가지 유형으로 분류한 기준은 다음과 같다.

COBOL, C++, Java 등 프로그래밍 언어에 관계 없이 소프트웨어는 속성(Attribute)과 함수(Function), 제어(Control) 데이타 저장(Data Storage)으로 구성된다. 현재 대표적인 컴포넌트 참조 모델인 CORBA, EJB, .NET 등은 객체 기술을 이용하고 있다. 따라서 대부분의 컴포넌트 내부는 클래스와 클래스들간의 관계로 구성 되어 있다.

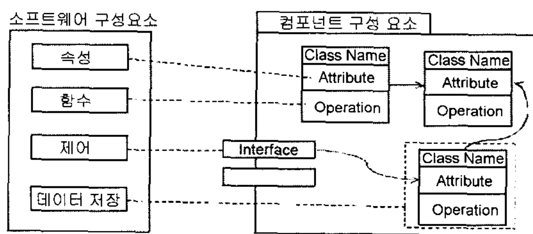

그림 2 가변성 유형 분류 근거

본 논문의 컴포넌트 가변성 유형 기준은 그림 2와 같이 소프트웨어의 구성 요소와 일대일로 맵핑되는 컴포넌트 구성요소를 기준으로 가변성 유형을 분류하였다. 이러한 분류는 도메인이나 개발 환경에 독립적이며 컴포넌트에 존재할 수 있는 모든 가변성을 분류할 수 있다.

소프트웨어 함수는 기능을 나타내는 요소로써, 컴포넌트 내부 클래스의 오퍼레이션과 맵핑된다. 오퍼레이션의 알고리즘과 로직이 변경 될 수 있는 부분으로 논리 가변성과 맵핑된다. 소프트웨어 속성은 상태를 나타내는 요소로써, 컴포넌트 내부 클래스의 속성과 맵핑 된다. 상태는 환경에 따라 변경 될 수 있는 것으로, 속성 가변성 유형에 해당된다.

소프트웨어 제어는 구성 요소들의 호출 순서를 관리하는 것으로써, 컴포넌트의 인터페이스를 통해서 클래스 오퍼레이션의 호출 순서를 제어 한다. 이러한 메시지 호출 순서는 변경 될 수 있으며, 워크플로우 가변성과 맵핑 된다. 소프트웨어의 데이타를 저장하기 위한 데이타베이스 테이블의 물리적 스키마는 컴포넌트 데이타 스키마와 다를 수 있다. 이러한 데이타베이스 테이블의 물리적 스키마는 컴포넌트의 데이타 스키마와 다를 수 있다. 이 부분은 데이타 영구 가변성과 맵핑 된다.

4.1 논리 가변성(Logic Variability)

논리 가변성은 여러 패밀리멤버에서 동일한 기능을 수행 하지만 그 기능을 수행함에 있어 다른 알고리즘을 사용하거나, 로직이 다를 경우 논리 가변성이라 한다. 논리 가변성의 단위는 컴포넌트 내부 클래스의 멤버 함수에 해당 된다. 한 컴포넌트내의 멤버 함수인 f()를 수행하는 알고리즘이나 로직이 패밀리 멤버마다 다른 경우를 의미한다.

(정의 9)의 *SetOfVariableRF*의 하나인 논리 가변성은 $G_i.logic(FamilyMem_j) \neq G_i.logic(FamilyMem_k)$ 조건을 만족한다. $G_i.logic$은 패밀리멤버의 요구사항 중 논리적인 측면을 나타내는 것으로써, 논리 가변성 유형에서 이를 상세하게 정의한다.

*SetOfVariableRF*의 패밀리멤버들의 $G_i.logic$을 알기 위해서는 요구사항들을 수집하여 패밀리멤버들간의 알고리즘이나 로직을 비교하거나 기능적, 동적, 정적 모델링을 하는 과정에서 알아 낼 수 있다. 이러한 패밀리멤버들의 로직이나 알고리즘을 알아내기 위한 지침이나 과정을 정형적인 함수로 표현 할 수 있다.

(정의 10) *FunctionOfLogic*은 패밀리멤버들의 로직이나 알고리즘을 찾는 함수

본 논문에서는 *FunctionOfLogic*을 찾아내는 기법에 대해서는 다루지 않는다. (정의 6)의 *CommonSetOfRF*에 포함되는 패밀리멤버들의 요구사항에 대응하는 알고리즘이 존재한다.

(정의 11) *SetOfALG*는 *CommonSetOfRF*에 포함되는 알고리즘의 집합

따라서, *FunctionOfLogic*은 다음과 같이 표현된다.

$$FunctionOfLogic : CommonSetOfRF \rightarrow P(SetOfALG),\quad P\text{는 멱집합}$$

하지만, 이 정의는 특정한 *FamilyMem*마다 *SetOfALG*가 다르므로 좀 더 상세한 정의가 필요하다. 따라서 *FunctionOfLogic*을 재 정의하면, *FamilyMem*의 *CommonSetOfRF* 원소인 *RF*에 대응하는 값, 즉 *SetOfALG*이 다르다.

$$FunctionOfLogic : CommonSetOfRF \quad Family \rightarrow P(SetOFALG)$$

따라서 한 패밀리에 속하는 서로 다른 두 개의 패밀리멤버들에 대하여, 요구사항들이 컴포넌트 공통성에 포함될 때, 논리 가변성은 다음과 같이 표현된다.

$\exists FamilyMem_i,\ FamilyMem_j(i \neq j) \in Family,$

$\exists RF \in CommonSetOfRF \Rightarrow FunctionOfLogic(RF_k,$

$FamilyMem_i) \neq FunctionOfLogic(RF_k, FamilyMem_j)$

위의 논리가변성의 수식은 동일한 요구사항이 패밀리 멤버에 따라 다른 알고리즘에 대응되는 경우 논리가변성으로 분류됨을 나타낸다.

그림 3은 $(RF_k,\ FamilyMem_i) \rightarrow \{alg_1,\ alg_2\}$, $(RF_k,\ FamilyMem_j) \rightarrow \{alg_1',\ alg_2'\}$로써 패밀리멤버마다 *SetOfALG*의 원소인 *alg*의 집합이 다름을 나타낸다.

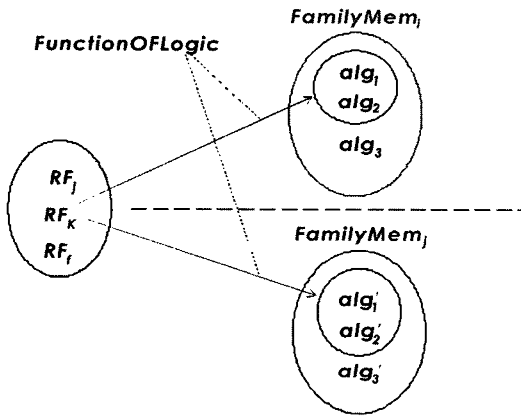

그림 3 가변적인 논리 예제

논리 가변성의 예로는 병원 패밀리의 A , B, C 병원에서 환자 등록이라는 동일한 기능을 수행 한다. 이때, A 병원은 시스템에 저장 되어있는 환자의 주민등록 번호를 이용하여, 그 값으로 환자에 대한 고유 아이디를 생성한다. B 병원은 등록한 환자에 대하여 해당 아이디를 생성할 당시의 날짜와 절대 시간으로 환자의 고유한 아이디를 생성한다. C 병원에서는 등록할 환자에 대하여 일련의 순차적인 고유 번호를 정한 후 그 번호를 이용하여 환자의 아이디를 생성한다.

이 세 병원에서 환자 등록이라는 기능은 같지만, 그 내부에서 환자 아이디를 생성하는 로직은 모두 다르다. 이런 경우 논리 가변성 유형으로 분류 된다. 본 논문에서는 다음의 코드처럼 논리 가변성의 예를 보여 준다.

여러 패밀리 멤버가 병원 관리 컴포넌트를 사용할 경우 논리 가변성이 요구사항 분석 단계에서 식별되어 컴포넌트에 반영 되지 않는다면, 세 병원이 모두 동일한

```
String registerPatient(String name, String ssn, int type){
}

// 주민등록번호로 ID 생성
String getIDusedBySSN(String ssn){
        return String.toString(Integer.parseInt(ssn) +  1);
}

//등록 날짜와 절대시간으로 ID 생성
String getIDByDate(){
        return String.toString( System.currentTimeMillis() );
}

//등록 순차번호로 ID 생성
String getIDBySequence(){
        return String.toString(count+ + );
}
```

방식으로 환자를 등록 해야만 한다. 환자 등록 로직은 각 병원의 정책으로 인하여 변경 될 수 있다. 따라서, 논리 가변성 유형은 패밀리멤버의 요구사항에 맞게 기능 수행 로직이나 알고리즘을 변경 할 수 있는 장치를 제공하여, 재사용성이 향상된다.

4.2 속성 가변성(Attribute Variability)

여러 패밀리멤버에서 동일한 기능을 수행 하지만, 필요로 하는 속성의 종류, 타입 또는 개수가 다른 경우에 속성 가변성이라 한다. 단, 속성의 타입이 호환 가능한 경우에는 동일한 속성으로 분류되어, 속성 가변성 유형에서 제외 된다. 속성 가변성의 단위는 객체 수준으로써, 객체의 속성에 해당 된다. 한 컴포넌트내의 멤버함수인 f( )를 수행하는 경우 필요로 하는 컴포넌트를 사용하는 사용자 마다 필요로 하는 속성이 다른 경우로, 필요로 하는 속성을 설정하는 것은 Required 인터페이스를 통해 설정된다.

(정의 9)의 *SetOfVariableRF*의 하나인 속성 가변성은 $G_i.attribute(FamilyMem_j) \neq G_i.attribute(FamilyMem_k)$ 조건을 만족한다. $G_i.attribute$은 패밀리멤버의 요구사항 중 속성측면을 나타내는 것으로써, 속성 가변성 유형에서 이를 상세하게 정의 한다.

*SetOfVariableRF*의 패밀리멤버들의 $G_i.attribute$을 알기 위해서는 요구사항들을 수집하여 패밀리멤버들간의 속성들의 타입이나 개수를 비교하거나 기능적, 동적, 정적 모델링을 하는 과정에서 알아 낼 수 있다. 이러한 패밀리멤버들의 속성을 알아내기 위한 지침이나 과정을 정형적인 함수로 표현 할 수 있다.

(정의 12) *FunctionOfAttribute*은 패밀리멤버들의 속성을 찾는 함수

본 논문에서는 *FunctionOfAttribute*을 찾아내는 기법에 대해서는 다루지 않는다. (정의 6)의 *CommonSetOfRF*에 포함되는 패밀리멤버들의 요구사항에 대응하는 속성

이 존재한다. 이 속성에는 속성의 타입이 있다. 속성타입이 패밀리멤버들간에 다른 경우 속성가변성으로 분류된다.

(정의 13) *Attr*은 패밀리멤버의 요구사항에 포함된 속성

(정의 14) *Attrtype*은 패밀리멤버 요구사항에 포함된 속성의 타입

(정의 15) *SetOfAttr*은 *CommonSetOfRF*에 포함되는 속성의 집합

따라서, *FunctionOfAttribute*은 다음과 같이 표현된다.

FunctionOfAttribute : *CommonSetOfRF*→ *P*(*SetOfAttr*), *P*는 멱집합

하지만, 이 정의는 특정한 *FamilyMem*마다 SetOfAttr이 다르고, 패밀리멤버간의 *Attrtype*의 호환여부를 검사하는 기능이 없으므로 좀 더 상세한 정의가 필요하다.

(정의 16) *FunctionOfTypeCast*는 속성 타입의 호환여부를 검사하는 함수

FunctionOfTypeCast : *Attrtype*×*Attrtype*→ *Boolean*

따라서 *FamilyMem*의 *CommonSetOfRF* 원소인 *RF* 에 대응하는 값, 즉 *SetOfAttr*이 다른 것을 찾고, 패밀리멤버들의 속성 타입 호환여부를 검사하는 기능을 포함한 함수를 정의 한다.어떤 속성의 타입이 다른 타입으로 형 변환(Type Casting)될 수 있다면, 프로그램 언어상에서 처리해 줄 수 있으므로 컴포넌트의 속성 가변성 유형에서 제외한다. 따라서, 다음의 *FunctionOfAttr2* 에서 $Attr_i$, $Attr_j$이 호환되면, 그 속성은 속성 가변성에서 제외 된다.

FunctionOfAttr1 : *CommonSetOfRF* *Family* → *P*(*SetOfAttr*)

FunctionOfAttr2 : *P*(*SetOfAttr*)→*P*(*SetOfAttr*)

$\forall Attr_i, Attr_j (i \neq j), Attr_i, Attr_j \in SetOfAttr$:

If FunctionOfTypeCast==true

Then *FunctionOfAttr1* −{$Attr_j$}

FunctionOfAttr=*FunctionOfAttr2* · *FunctionOfAttr1*

따라서 한 패밀리에 속하는 서로 다른 두 개의 패밀리멤버들에 대하여, 요구사항들이 컴포넌트 공통성에 포함될 때, 속성 가변성은 다음과 같이 표현 된다.

$\exists FamilyMem_i, FamilyMem_j (i \neq j) \in Family,$

$\exists RF \in CommonSetOfRF \Rightarrow FunctionOfAttr(RF_k, FamilyMem_i) \neq FunctionOfAttr(RF_k, FamilyMem_j)$

위의 속성가변성의 수식은 동일한 요구사항이 패밀리멤버에 따라 다른 종류의 속성들과 다른 타입의 속성들에 대응되는 경우 속성가변성으로 분류됨을 나타낸다. 그림 3은 $(RF_k, FamilyMem_i) \rightarrow \{attr_1, attr_2, attr_3\}$, $(RF_k, FamilyMem_j) \rightarrow \{attr_1, attr_3, attr_5, attr_7\}$로써 패밀리멤버마다 *SetOfAttr*의 원소인 *Attr*의 집합이 다름을 나타낸다.

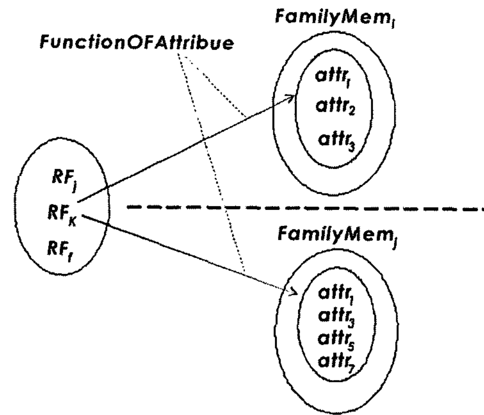

그림 4 가변적인 속성 예제

속성 가변성의 예로는 표 1처럼 A 은행과 B 은행 C 은행이 계좌 개설 이라는 기능을 수행 하는 경우 각 은행에 필요한 속성의 개수와 종류 및 타입이 다른 경우이다. A 은행에서 필요로 하는 속성은 상담원 정보, 현금, 수수료, 인출 한도, 패스워드, 예금 과목, 계좌 번호이다. B은행에서는 상담원 정보, 수수료, 패스워드, 예금 과목, 계좌번호이다. C 은행에서는 현금, 인출 한도, 패스워드, 예금 과목, 계좌번호이다. 따라서, 상담원 정보, 현금, 수수료, 인출 한도는 가변 속성이다. 이 세 은행에서 계좌를 개설 하는데 필요한 속성의 타입과 그 종류가 모두 다르다. 이런 경우 속성 가변성 유형으로 분류된다.

표 1에서 수수료 속성은 가변 속성이다. 은행 A와 은행 B는 계좌를 개설 할 경우 수수료를 받도록 되어 있지만, 은행 C에서는 계좌 개설은 고객에 대한 서비스로써, 수수료를 받지 않는다. 만약 속성 가변성이 식별되지 않고, 은행 C의 계좌 등록 기능이 기본으로 사용 된다면, 은행 A와 은행 B가 여신 컴포넌트를 사용할 경우 계좌 개설 수수료를 청구 할 수 없다. 따라서, 속성 가변성 유형 기준에 따라 식별된 속성이 요구 사항 분석 단계에서 분류 되어 각 은행의 정책에 맞도록 컴포넌트 상태를 커스터마이제이션 할 수 있어, 재사용성이 높은 컴포넌트를 개발할 수 있다.

표 1 속성 가변성 예제

Category	Function Name	Attribute Name	Bank A		Bank B		Bank C	
			Uses	Type	Uses	Type	Uses	Type
여신	계좌 계설	상담원정보	○	Text	○	Text		
		현금	○	Num			○	Num
		수수료	○	Num	○	Num		
		인출한도	○	Num			○	Num
		패스워드	○	Text, Num	○	Num	○	Text, Num
		예금과목	○	Text	○	Text	○	Text
		계좌번호	○	Num	○	Text	○	Num

4.3 워크플로우 가변성(Workflow Variability)

워크플로우 가변성은 여러 어플리케이션에서 동일한 목적을 수행 하지만 그 목적을 수행하기 위한 내부 메시지의 흐름이 가변적인 경우를 말한다. 워크플로우 가변성은 컨트롤러 수준으로써 컴포넌트 단위이다. 컨트롤 수준이란 작업 목적을 수행하기 위해 어떠한 명령을 호출 했을 경우, 그 명령을 수행하기 위해 필요한 여러 종류의 멤버 함수를 호출하고 그 호출 순서의 흐름을 관리 하는 것을 말한다.

한 컴포넌트의 인터페이스를 수행할 때 요구되는 메시지 흐름의 가변성으로서, 즉 메시지 Op_i를 실행했을 경우 한 회사에서는 $Obj_i.f()$; $Obj_7.k()$; 순서로 호출되는 반면, 다른 회사는 $Obj_i.f()$; $Obj_8.k()$;가 호출되는 경우이다. $Obj_i.fj()$ 메시지의 i, j가 1≤i, j≤n 인 경우 A 회사의 i, j와 B 회사의 i, j의 값이 다른 경우 워크플로우 가변성이다.

워크플로우 가변성 수행시 특정 조건에 의해서만 수행 되는 경우가 있다. 이 경우 특정 조건이 하나의 패밀리 멤버에서만 존재 하거나, 조건문에 의해서 흐름이 변경된다면, 워크플로우 가변성으로 분류 될 수 없다. 워크플로우가변성의 설정은 Required 인터페이스에 의해 설정 된다.

(정의 9)의 SetOfVariableRF의 하나인 워크플로우 가변성은 $G_i.workflow(FamilyMem_j) \neq G_i.workflow(FamilyMem_k)$ 조건을 만족한다. $G_i.workflow$는 패밀리멤버의 요구사항 중 워크플로우를 나타내는 것으로써, 워크플로우 가변성 유형에서 이를 상세하게 정의 한다.

*SetOfVariableRF*의 패밀리멤버들의 $G_i.workflow$를 알기 위해서는 요구사항들을 수집하여 패밀리멤버들간의 워크플로우를 비교하거나 기능적, 동적, 정적 모델링을 하는 과정에서 알아 낼 수 있다. 이러한 패밀리멤버들의 워크플로우를 알아내기 위한 지침이나 과정을 정형적인 함수로 표현 할 수 있다.

(정의 17) *FunctionOfWorkflow*는 패밀리멤버들의 워크플로우를 찾는 함수

본 논문에서는 *FunctionOfWorkflow*을 찾아내는 기법에 대해서는 다루지 않는다. (정의 6)의 *CommonSetOfRF*에 포함되는 패밀리멤버들의 요구사항에 대응하는 워크플로우가 존재한다. 워크플로우 내에는 워크플로우를 구성하는 멤버 함수들이 존재 한다.

(정의 18) OP는 한 워크플로우를 구성하는 멤버함수

(정의 19) *WF*는 *CommonSetOfRF*에 포함되는 워크플로우라 하며, 각 *WF*는 여러 메시지의 전송 순서를 가진다. 즉, $WF=\{op_1, op_2, op_3, \cdots\cdots, op_n\}$로서, 멤버함수는 워크플로우에 포함된다.

*FunctionOfWorkflow*는 다음과 같이 표현된다.

$$FunctionOfWorkflow : CommonSetOfRF \rightarrow P(WF),$$

P는 멱집합

하지만, 이 정의는 특정한 *FamilyMem* 마다 *WF*가 다르므로 좀 더 상세한 정의가 필요하다. 따라서 *FunctionOfWorkflow*을 재 정의하면, *FamilyMem*의 *CommonSetOfRF* 원소인 *RF*에 대응하는 값, 즉 *WF*가 다르다.

$$FunctionOfLogic : CommonSetOfRF \times Family \rightarrow P(WF)$$

따라서 한 패밀리에 속하는 서로 다른 두 개의 패밀리멤버들에 대하여, 요구사항들이 컴포넌트 공통성에 포함될 때, 워크플로우 가변성은 다음과 같이 표현 된다.

$$\exists FamilyMem_i, FamilyMem_j(i \neq j) \in Family, \exists RF \in CommonSetOfRF \Rightarrow FunctionOfWorkflow(RF_k, FamilyMem_i) \neq FunctionOfWorkflow(RF_k, FamilyMem_j)$$

위의 워크플로우 가변성의 수식은 동일한 요구사항이 패밀리멤버에 따라 다른 워크플로우에 대응되는 경우 워크플로우 가변성으로 분류됨을 나타낸다. 그림 3은 $(RF_k, FamilyMem_i) \rightarrow WF$, $(RF_k, FamilyMem_j) \rightarrow WF'$로써 패밀리멤버마다 *WF*의 원소인 *OPi*의 개수와 순서가 다름을 나타낸다.

예를 들면 은행 업무에서 대출 업무 중 기업 자금 대출 목적의 기능을 수행 할 경우 A 은행에서는 기업 자금 대출 승인을 받기 위해서, 행원으로부터 책임자와 지점장의 승인을 받는다. B 은행에서는 행원과 지점장의 승인을 받는다. C 은행에서는 행원, 책임, 지점장, 본부장의 승인을 받는다. 이럴 경우 기업 자금 대출 목적을 수행 하는 동안에 멤버 함수 호출 순서 흐름과 호출되는 멤버함수의 개수가 다른 것을 알 수 있다. 이런 경우 워크플로우 가변성 유형으로 분류 된다.

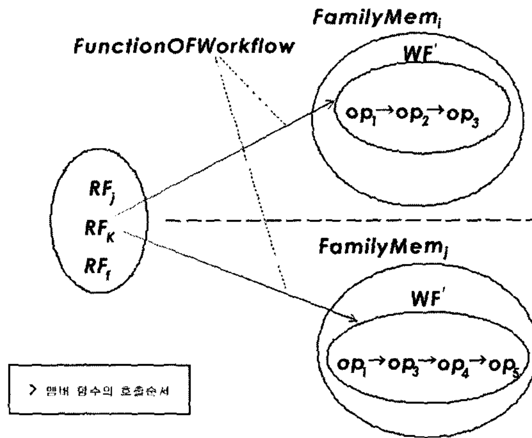

그림 5 가변적인 워크플로우 예제

그림 6과 같이 각 은행들은 기업 자금 대출을 하기 위해서 서로 다른 메시지의 흐름을 가지고 있다. 요구사항 분석 단계에서 각 은행의 업무 처리를 위한 워크플로우 가변성이 식별 되지 않는다면, 한 은행의 업무 흐름 방식을 따라야 한다. 즉 컴포넌트 내부에 작업 흐름이 고정 되어 있어, 컴포넌트를 이용하는 회사에서 컴포넌트에 내장된 작업흐름에 맡도록 업무 순서를 바꾸지 않는다면, 컴포넌트를 재사용 할 수 없다. 워크플로우 가변성은 각 패밀리 멤버에 맞는 작업 흐름 가변성을 내장하여, 컴포넌트 재사용성을 향상 시킨다.

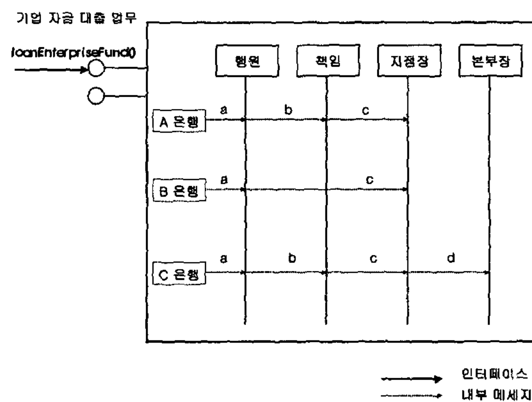

그림 6 워크플로우 가변성 예제

4.4 데이타 영구 가변성(Data Persistent Variability)

컴포넌트의 데이타 영구 스키마가 어플리케이션의 데이타베이스 스키마와 물리적으로 다를 경우 데이타 영구 가변성이라 한다. 여러 어플리케이션에서 컴포넌트를 재사용하기 위하여 컴포넌트 내부의 데이타 영구 스키마는 각각의 어플리케이션의 데이타베이스 스키마에 논리적으로 포함되거나 같아야 한다.

컴포넌트의 데이타 영구 스키마는 어플리케이션의 데이타베이스 스키마와 물리적으로 다를 수 있다. 어플리케이션의 데이타베이스가 관계형 데이타베이스일 경우, 데이타베이스 스키마는 테이블로 표현된다. 기존의 어플리케이션의 데이타베이스의 테이블 스키마가 컴포넌트 데이타 영구스키마의 표현 양식인 클래스 다이어그램과 일대일로 매핑 되지 않는 경우가 많다. 따라서 컴포넌트의 데이타 영구 스키마를 어플리케이션의 데이타베이스 스키마에 맞추어 커스터마이제이션 해야 한다.

데이타 영구 가변성은 기존의 어플리케이션 시스템의 데이타베이스 스키마와 컴포넌트의 스키마가 다른 경우에 발생하거나, 타 시스템의 데이타베이스 스키마와 컴포넌트가 연동 할 경우에 발생 한다.

컴포넌트 데이타 영구 스키마는 컴포넌트에서 사용되는 데이타 영구 데이타의 스키마를 정의한 것이다. 컴포넌트 데이타 영구스키마는 컴포넌트 내부의 클래스를 통해 표현 되거나 컴포넌트 스키마를 표현할 수 있는 컴포넌트 모델을 통해 정의 된다.

(정의 20) *ComponentSchma*는 한 컴포넌트에서 사용되는 데이타 영구 데이타 스키마

(정의 21) *SetOfComponentSchma*는 컴포넌트 데이타 영구 데이타 스키마의 집합

$ComponentSchma \in SetOfComponentSchma$

$SetOfComponentSchma=\{ComponentSchma_i \mid 1 \leq i \leq n$, n은 컴포넌트 데이타 영구 스키마의 개수$\}$

*ComponentSchma*는 한 개의 클래스 또는 두 개 이상의 클래스들의 속성으로 구성 된다.

(정의 22) $SetAttributeOfClass_j$는 컴포넌트 스키마를 구성하는 클래스의 속성 집합, j는 어떤 클래스.

따라서 $SetAttributeOfClass_j=\{attr_i \mid 1 \leq i \leq n$, n은 속성들의 개수$\}$

$ComponentSchma=\{SetAttributeOfClass_k\}$

컴포넌트 스키마를 물리적으로 표현하면 다음과 같다.

$ComponentSchema=SetAttributeOfClass_j \cup SetAttribueOfClass_k$

컴포넌트를 사용하는 회사에는 기존 어플리케이션의 데이타베이스 스키마가 존재한다. 기존 어플리케이션의 데이타베이스 스키마는 테이블로 구성 된다.

(정의 23) SetColumnOfTablei는 어플리케이션의 데이타베이스의 하나의 테이블 스키마, I는 어떤 테이블

(정의 24) *SetTableOfCompany*는 어플리케이션의 데이타베이스 테이블 스키마 집합

하나의 *SetColumnOfTable*은 여러 열(*Column*)로 구성된다.

(정의 25) *SetOfColumn*은 어플리케이션의 데이타베이스에 존재하는 모든 열의 집합

$SetColumnOfTable_j = \{column_i \mid 1 \le i \le n$, n은 열의 개수$\}$

$SetTableOfCompany = SetColumnOfTable_j \cup SetColumnOfTable_k$

*ComponentSchema*는 하나 또는 여러 개의 *SetColumnOfTable*의 집합과 매핑된다. 이때, *ComponentSchema*를 구성하는 각각의 *SetAttributeOfClass*와 *SetTableOfCompany*을 구성하는 하나의 *SetColumnOfTable*를 매핑하는 함수로 표현 할 수 있다.

(정의 26) *Store*는 *SetAttributeOfClass*와 *SetColumnOfTable*을 매핑하는 함수

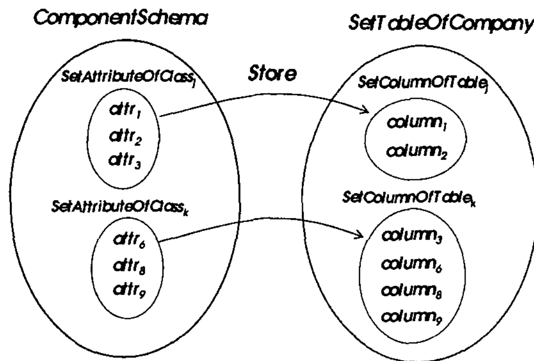

그림 7 가변적인 데이타 영구 예제

이때 *ComponentSchema*를 구성하는 각각의 *SetAttributeOfClass*가 *SetTableOfCompany*을 구성하는 하나의 *SetColumnOfTable*와 1 : 1로 매핑 되는 경우는 ComponentSchema를 커스터마이제이션 없이 이용할 수 있으므로 데이타 영구 가변성이 아니다. 정형화하여 표현하면 다음과 같다.

$SetColumnOfTable \subset P(SetOfColumn)$

$Store(SetAttributeOfClass) \subset SetColumnOfTable \wedge$

$(Store(SetAttributeOfClass) - \text{SetColumnOfTable} = \emptyset)$

*ComponentSchema*를 커스터마이제이션해야 하는 데이타 영구 가변성인 경우는 두 가지 경우가 있다. 첫번째, *SetAttributeOfClass*가 하나의 *SetColumnOfTable*에 포함되지만, *SetColumnOfTable*에 열이 추가로 있는 경우이다.

$Store(SetAttributeOfClass) \subset SetColumnOfTable \wedge \sim$

$(Store(SetAttributeOfClass) \subset SetColumnOfTable \neq \emptyset)$

두번째는 *SetAttributeOfClass*가 두개 이상의 *SetColumnOfTable*에 포함되는 경우이다.

$Store(SetAttributeOfClass) \subset P(SetOfColumn) \wedge \sim$

$(Store(SetAttributeOfClass) \subset SetColumnOfTable)$

그림 8은 A 병원과 B병원에서 진료 컴포넌트를 사용할 경우에 발생하는 데이타 영구 가변성이다. 진료 컴포넌트 데이타 영구 스키마의 환자 클래스의 서브클래스인 응급환자 클래스는 A병원의 진료 관련 데이타베이스 스키마의 응급환자 테이블과 매핑된다. 이때, 응급환자 테이블의 입원날짜(enterDate)열이 추가 되어 있다. 따라서, *SetAttributeOfClass*가 하나의 *SetColumnOfTable*에 포함되지만, *SetColumnOfTable*에 열이 추가로 있는 경우의 데이타 영구 가변성이다.

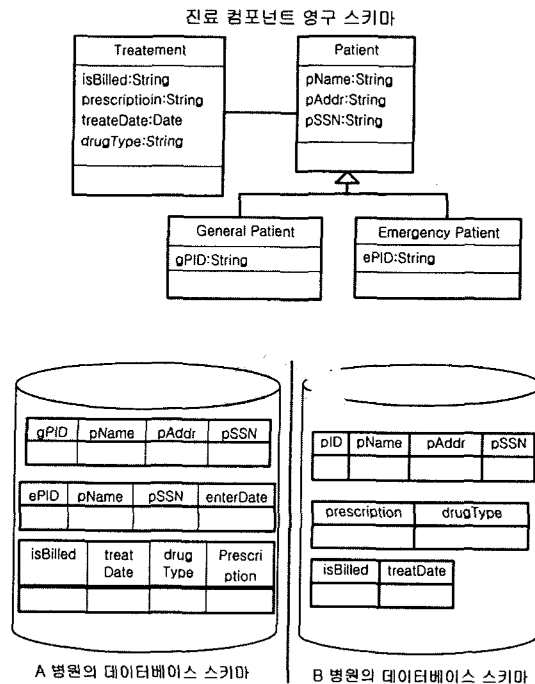

그림 8 데이타 영구가변성 예제

진료 컴포넌트 데이타 영구 스키마의 진료 테이블이 B병원에서 두개의 진료 테이블로 물리적으로 나뉘어 매

핑 되고 있다. 따라서, *SetAttributeOfClass*가 두 개 이상의 *SetColumnOfTable*에 포함되는 경우의 데이타 영구 가변성이다. 진료 컴포넌트를 사용하기 위하여 컴포넌트의 데이타 영구 스키마를 병원의 데이타베이스의 스키마에 맞게 커스터마이제이션 해야 한다.

5. 컴포넌트 가변성의 SCOPE

본 논문에서는 컴포넌트 내에서 가변성의 유형에 관계없이 존재 할 수 있는 가변성의 경우의 수를 컴포넌트 가변성의 Scope라고 정의한다. 가변성 Scope는 패밀리 멤버들의 요구사항에 존재하는 컴포넌트 가변성의 개수를 결정해야 하는 중요한 문제이다.

만약 컴포넌트 가변성의 경우의 수가 많다면 컴포넌트 내부에서 포함해야 하는 가변성의 개수가 많아져, 여러 패밀리 멤버에서 자신의 환경에 맞도록 컴포넌트를 커스터마이제션 할 수 있어 컴포넌트의 재사용성이 향상 된다. 하지만 컴포넌트 가변성의 개수가 많을수록 컴포넌트 내부가 복잡해지고, 개발 비용 또한 상승하게 된다. 따라서, 컴포넌트 가변성의 경우의 수를 결정할 때에는 도메인 전문가, 분석가, 개발자, 사용자들이 참여하여 다음과 같은 지침을 통한 가변성 Scope를 결정한다.

특정 패밀리 멤버를 지원하기 위하여 컴포넌트 가변성을 포함해야 하는가?

패밀리 멤버 요구사항 분석을 통해 추출된 컴포넌트 가변성이 꼭 필요한 가변성인가?

식별된 가변성이 컴포넌트 참조 모델에서 직접적으로 지원할 수 있는가?

본 논문에서는 컴포넌트 가변성 범위를 여러 패밀리의 요구사항 분석 단계에서 식별되는 가변성의 경우의 수를 이진 Scope, 선택 Scope로 나누고, 추후에 가변성으로 추출 될 수 있으나 그 형태를 정확히 알 수 없는 가변성의 경우의 수를 Unknown Scope로 구분한다. 컴포넌트 가변성 Scope에 따라, 가변성이 발생할 수 있는 경우의 수를 결정 함으로써, 컴포넌트 내부에 가변성의 포함 여부를 결정하고, 컴포넌트 가변성 설계시에 가변성 개수를 확인하고 각 가변성에 대한 설계를 예측 할 수 있다.

5.1 이진 Scope(Binary Scope)

이진 범위는 컴포넌트 내에 가변성 유형에 관계없이 가변성의 경우의 수가 두 가지인 경우이다. 분석 단계에서 추출된 가변성의 선택 범위가 두 가지인 경우로 가변성의 개수가 명확한 경우이다. 이진 범위는 컴포넌트 가변성의 최소의 수로써, 컴포넌트 내부에 기본으로 실행되는 공통 기능 이외에 하나의 가변성을 선택할 수 있도록 한 것이다. 이진 Scope는 컴포넌트 내부에 설계할 가변성의 경우의 수가 두 개로 제한되어 있어. 개발 비용이 줄어 들지만, 여러 패밀리 멤버에서 커스터마이제이션하여 사용될 수 없다는 단점이 있다.

이진 범위 가변성의 예를 들면, 이자 계산을 하는 확정 이자 계산 방식과 예상 이자 계산 방식의 두 가지 가변성이 존재 하는 경우이다. 그림 9에서 Required 인터페이스를 통해 이진 Scope에서 이자계산 방식을 확정 이자 계산 방식으로 설정하면, 이자 관리 컴포넌트는 확정이자 방식으로 실행 된다.

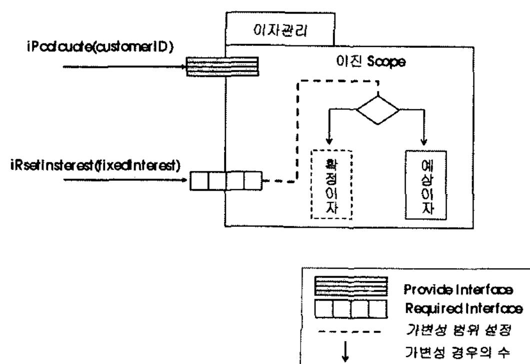

그림 9. 이진 Scope 예제

이진 범위 가변성은 컴포넌트 가변성 분석단계에서 식별된 각 유형들이 가지는 경우의 수를 확인 함으로써, 설계시에 대안을 마련할 수 있다. 이진 범위는 컴포넌트 설계시에, 선택 조건문으로 설계된다.

5.2 선택 Scope(Selection Scope)

선택 Scope 가변성은 컴포넌트 내에 가변성의 경우의 수가 여러 개 존재하며, 가변성의 경우의 수를 예측할 수 있는 경우이다. 선택 Scope는 그 경우의 수가 3개 이상인 경우이다. 패밀리 멤버의 요구사항 중 기업 대출 승인이라는 공통 기능을 수행 하는 중에, 대출 승인을 받기 위한 결제 흐름이 다르다면, 각 흐름은 워크플로우 가변성으로 추출 된다. 워크플로우 가변성 중 세개 이상의 가변성의 경우의 수를 컴포넌트 내부에 설계하기로 결정한다면, 선택 Scope에 해당된다.

그림 10은 기업 대출 관리 승인에 대해 세가지 경우의 워크플로우 가변성이 있는 경우이다. A 경우는 기업 자금 대출 승인을 받기 위해서, 행원으로부터 책임자와 지점장의 승인을 받는 흐름이다. B 경우는 행원과 지점장의 승인을 받는 흐름이다. C 경우는 행원, 책임, 지점장, 본부장의 승인을 받는 흐름이다. Required 인터페이

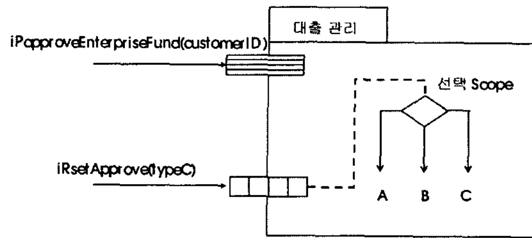

그림 10 선택 Scope 예제

스를 통해 C를 설정함으로써, 행원, 책임, 지점장, 본부장의 승인을 받는 흐름이 결정 된다.

선택 Scope는 컴포넌트 내부에 가변성의 경우의 수를 여러 개 포함하여 설계됨으로써, 여러 패밀리 멤버에서 커스터마이제션하여 사용 할 수 있는 장점이 있다. 하지만, 컴포넌트 내부 설계가 복잡해지고, 개발 비용이 증가한다.

5.3 Unknown Scope

Unknown Scope는 선택 Scope중에서 선택 경우의 수가 추가 될 가능성이 있는 경우이다. 이런 경우는 컴포넌트 설계 시에 가변성의 경우의 수를 정확하게 예측할 수 없으며, 추후에 회사의 정책이나 업무 흐름, 기능, 데이타 등이 추가 될 것을 예측 할 수 있을 때, 컴포넌트 커스터마이제이션을 지원하도록 설계 된다.

Unknown Scope가 존재하는 경우 컴포넌트 설계시에 추가될 가변성을 포함 할 수 있도록 장치를 제공해 주어야 한다. Unknown Scope는 컴포넌트 설계시에, 추가 될 가변성의 경우의 수를 예측할 수 없으므로, 가변성 추가가 가능하도록 Plug-in 방식과 같은 설계 기법을 제공해야 한다.

그림 11의 점선 화살표로 표시된 부분은 회사 정책상 기업대출관리 승인의 흐름이 추가 될 경우를 예측하여 Unknown Scope가 선택된 경우이다.

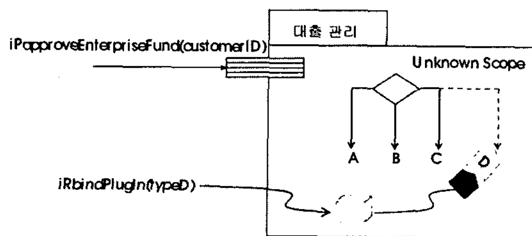

그림 11 Unknown Scope 예제

6. 평가

소프트웨어 재사용성 문제를 해결 할 수 있는 기술인 컴포넌트가 한 회사 또는 한 어플리케이션에서 제한적으로 재사용되어 왔으며, 결과적으로 유지보수 향상 목적의 컴포넌트가 개발 되어 왔다.

이러한 한계를 극복하고자 여러 회사 또는 어플리케이션에서 컴포넌트를 재사용하고 조립 할 수 있는 컴포넌트를 개발하기 위하여 컴포넌트 가변성에 관한 연구의 필요성이 요구 되어 왔다. 하지만 컴포넌트 가변성과 컴포넌트 가변성 유형에 대한 명확한 정의가 제시 되지 않아 컴포넌트 가변성에 대한 의견을 주고 받거나 관점을 공유 할 수 없었다. 따라서 컴포넌트 가변성을 설계하고 각 환경에 맞게 커스터마이제이션 하여 사용 할 수 있는 장치나 기법에 관한 연구가 진행되지 못하였다.

본 논문은 소프트웨어 Product-Line 개념을 도입하여 도메인내에 포함된 여러 패밀리 멤버의 요구사항을 수집하여 공통 요구사항을 반영한 컴포넌트 공통성 내에서 컴포넌트 가변성 을 정의 하였다. 따라서 여러 패밀리 멤버에서 컴포넌트를 재사용 할 수 있으며, 각 환경에 맞도록 가변성을 커스터마이제이션하여 재사용성을 향상 시킬 수 있도록 하였다.

또한 소프트웨어를 구성하는 속성, 함수, 제어, 데이타 저장을 컴포넌트를 구성하는 요소들과 일대일로 매핑하여 컴포넌트에 존재하는 가변성을 속성, 논리, 워크플로우, 데이타 영구 가변성으로 분류하였다. 이 분류 기준은 도메인이나 환경에 독립적이며 모든 가변성들을 포함 할 수 있다. 네 가지 가변성 유형을 정형적으로 명세하여 그 기준을 정확하고 완전하게 정의하였으며, 각 유형 마다 그 예제를 소개하여 가변성 유형을 이해 할 수 있도록 하였다.

그림 12은 여러 패밀리멤버가 컴포넌트를 재사용 할 수 있도록 컴포넌트 가변성 모델을 설계하기 위하여 필요한 절차를 표현 한 것이다. 점선으로 표현된 부분은 본 논문에서 제시한 정의와 기법들이다.

그림 12는 여러 패밀리 멤버에서 재사용 될 수 있는 컴포넌트를 설계하기 위한 선행 작업들을 보여 주고 있다. 본 논문에서 제시한 컴포넌트 가변성 유형을 통하여

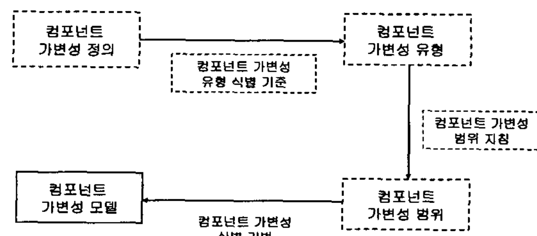

그림 12 컴포넌트 가변성 설계 절차

컴포넌트 가변성을 분석단계에서 식별할 수 있으며, 식별된 가변성은 컴포넌트 가변성 Scope에 따라 컴포넌트 내부에 컴포넌트 가변성의 포함 여부를 결정 할 수 있도록 하였다. 컴포넌트 가변성 유형을 기반으로 컴포넌트 설계시에 가변성을 표현할 수 있는 장치를 고안 할 수 있으며, 컴포넌트 커스터마이제이션 장치를 연구 할 수 있다.

컴포넌트 가변성 Scope를 이진 Scope와 선택 Scope 그리고 Unknown Scope로 분류 하였다. 컴포넌트 가변성 Scope를 제시함으로써, 컴포넌트 내부에 포함할 가변성의 개수를 결정 할 수 있다. 또한 컴포넌트 개발자들이 가변성이 발생할 수 있는 경우의 수를 예측하여 가변성을 분석 하고, 추출 할 때 컴포넌트 가변성을 컴포넌트 모델링에 상세하게 적용할 수 있다. 또한 커스터마이제이션을 수행할 때 변경 할 부분과 그 개수를 확인 할 수 있다.

Bachmann에서 제안하고 있는 가변성은 소프트웨어 아키텍쳐를 구성하고 있는 제품들 사이의 차이점이다. 그림 13은 Bachmann 기법에서 아키텍처에 존재하는 가변성과 컴포넌트 가변성이 존재하는 부분을 비교 한다. Bachmann 기법은 가변성을 찾는 대상이 제품 1과 제품 2를 비교하여 서로 다른 모듈인 B, D가 가변성으로 식별 된다. 하지만, 컴포넌트 가변성은 패밀리 A, B, C를 비교하여 서로 다른 부분을 가변성이라 하지 않고, 패밀리 멤버의 공통 요구사항을 찾아내고 그 부분에서 변경 될 수 있는 부분이 가변성 이다. 따라서 가변성의 정의와 가변성을 찾는 비교 대상이 다르다.

컴포넌트 가변성 유형과 비교 할 수 있는 Bachmann 에서 제시한 가변성 발생 부분이다. 첫째, 기능 부분이다. Bachmann은 소프트웨어 제품 내에 특정 기능의 존재 유무에 대한 가변성을 의미한다. 표 2는 Bachman에서 의미하는 기능 가변성의 예제로써, 자동차 A, B, C에 Car Radio 장치와 항법 시스템 장치의 존재 여부가 가변성 원인이다.

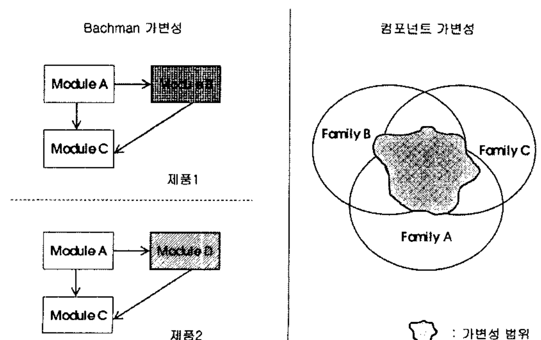

그림 13 Bachmann의 가변성과 컴포넌트 가변성의 추출 대상 비교

표 2 Bachmann의 기능 가변성 예제

	Car A	Car B	Car C
Car Radio			
Navigation system			

하지만 본 논문에서 제안한 논리 가변성(Logic Variability)은, 여러 패밀리 멤버에서 동일한 기능을 수행하지만 그 기능을 수행하기 위하여 서로 다른 알고리즘을 사용하거나, 로직이 다른 경우이다. 표 3은 병원 A, B, C에서 환자등록이라는 동일한 기능을 수행하는데 있어 환자 아이디 생성을 위한 알고리즘이 다른 경우를 보여주는 논리 가변성 예제이다. 따라서 논리 가변성 유형과 Bachmann의 기능 가변성이 발생하는 부분과는 의미가 다르다.

표 3 논리 가변성 예제

기능명	병원 A	병원 B	병원 C
register Patient( ) 환자등록	get ID used By SSN( ) 주민등록번호로 ID 생성	get ID By Date( ) 등록날짜와 절대시간으로 ID 생성	get ID By Sequence( ) 등록 순차 번호로 ID 생성

둘째, Bachman 기법에서 가변성이 발견되는 데이타 부분은 특정 자료구조가 제품 마다 다른 경우이다. 즉, 데이타 타입이 호환되지 않으면 가변성 원인이다. 하지만, 컴포넌트 속성 가변성의 정의는 여러 패밀리 멤버에서 동일한 기능을 수행하지만, 필요로 하는 속성의 종류, 타입 또는 개수가 다른 경우 이다. 단, 속성의 타입이 호환 가능한 경우에는 동일한 속성으로 분류되어, 속성 가변성 유형에서 제외 된다. 따라서, Bachmann에서 제안하고 있는 데이타 가변 부분은 본 논문에서 제안한 속성 가변성과는 다른 가변성이다.

셋째, Bachmann 기법에서 제안하고 있는 제어흐름 가변 부분은 데이타 변경의 일관성을 위한 Notification 장치와 같은 시스템 레벨에서의 제어 흐름 패턴을 의미한다. 컴포넌트 워크플로우 가변성은 여러 패밀리 멤버에서 동일한 목적을 수행 하지만 그 목적을 수행하기 위한 내부 메시지의 흐름이 가변적인 경우이다. 즉 업무를 처리하기 위한 작업 흐름의 가변성을 다루는 것으로써, 상위 단계의 메시지 흐름이 가변적인 경우이다. 따라서 Bachmann기법의 제어 흐름 가변 부분은 컴포넌

트의 워크 플로우 가변성과 그 목적이 다른 가변성이다. 따라서 Bachman에서 제안하고 있는 가변성 원인과 본 논문에서 제안한 4가지의 컴포넌트 가변성 유형과는 가변성이 존재하는 범위 뿐만 아니라 정의도 다르다.

표 4는 본 논문에서 제시한 컴포넌트 가변성 유형과 기존의 기법들이 제공하는 가변성의 유형들을 비교하고 있다. 기호 ×는 제안된 컴포넌트 가변성 유형이 존재하지 않는 경우이다. COMO는 소프트웨어 공학의 가변성 정의인 "어플리케이션에서 변경될 수 있는 부분"을 컴포넌트 가변성으로 정의하고 있다. 또한, 가변성 유형인 논리, 속성, 워크플로우 가변성에 대한 표현만 했을 뿐 명확한 정의나 설명을 표현하지 않고 있으며, 본 논문에서 제안한 컴포넌트 가변성 정의와 가변성 유형과는 다르다.

표 4 기존 기법과 컴포넌트 가변성 유형 차이점

기법 / 가변성유형	KobrA	FAST	COMO	Bachman	FODA	제안된 기법
논리가변성	×	×	×	×	×	○
속성 가변성	×	×	×	×	×	○
워크플로우가변성	×	×	×	×	×	○
데이타영구가변성	×	×	×	×	×	○

× : 존재하지 않음, ○ : 존재함

표 5는 본 논문에서 제시한 컴포넌트 가변성 Scope와 기존의 기법들이 제공하는 가변성의 Scope를 비교하고 있다. 컴포넌트 가변성 Scope는 가변성 유형별로 식별된 가변성을 컴포넌트 내부에 포함 시킬 경우의 수를 결정하고 확인하는 것으로써, 컴포넌트의 커스터마이제이션이 가능하도록 설계 될 수 있는 가변성 경우의 수를 결정 한다.

표 5 기존 기법과 컴포넌트 가변성 Scope 차이점

기법 / 가변성유형	KobrA	FAST	COMO	Bachman	FODA	제안된 기법
이진 Scope	×	×	×	×	×	○
선택 Scope	×	×	×	×	×	○
Unknown Scope	×	×	×	×	×	○

× : 존재하지 않음, ○ : 존재함

7. 맺음말

본 논문에서는 컴포넌트의 주 목적인 재사용성을 향상 시키기 위하여, 여러 패밀리 멤버에서 재사용할 수 있는 컴포넌트를 개발할 수 있도록 정형적으로 컴포넌트 가변성을 정의 하였다. 또한 컴포넌트의 특징을 반영한 컴포넌트 가변성을 정의하여, 컴포넌트 가변성 유형을 식별할 수 있는 기준을 제시하였다.

기존의 개략적으로 정의된 논리가변성 이외에 속성, 워크플로우, 데이타 영구 가변성을 추가 하여 도메인이나 개발 환경에 독립적이며, 컴포넌트에 존재하는 모든 가변성을 포함 하는 가변성 유형을 제시하였다. 또한 컴포넌트 가변성 유형을 정형적으로 정의 하여, 컴포넌트가을 식별할 수 있는 기준을 제시하였다.

컴포넌트 가변성의 Scope를 이진 Scope와 선택 Scope 그리고 Unknown Scope로 분류하여, 컴포넌트 내부에 포함될 컴포넌트 가변성 개수를 결정 할 수 있으며, 컴포넌트 내부에 포함될 가변성의 Scope 마다 커스터마이제이션 기법을 개발 할 수 있다.

본 논문에서 제시한 정형적인 컴포넌트 가변성 유형과 컴포넌트 가변성 Scope를 적용하면 도메인의 여러 패밀리 멤버들이 자신의 환경에 맞게 커스터마이제이션 하여 사용 할 수 있는 재사용성이 높은 고품질의 컴포넌트를 제공 할 수 있다.

참 고 문 헌

[1] D'Souza D., Wills A., *Objects, Components, and Frameworks with UML,* Addison Wesley, 1999.

[2] HP Company, *Engineering Process Summary: Fusion2.0,* Hewlett-Packard Company, January 1998.

[3] Cheesman J., Daniels J. *UML Components : A Simple Process for Specifying Component-Based Software,* Addison Wesley, 2001.

[4] Grahn G., "Transition from Conventional to Component-based Development", in Int'l Workshop on Component-Based Software Engineering, pp. 78-82, 1999.

[5] Veryad R., SCPIO: *Aims, Principles and Structure,* SCPIO Consortium, April 1998.

[6] Compuware Corp., *UNIFACE Development Methodology V7.2,* COMPUWARE Corp., 1998.

[7] Atkinson C., Bayer J., Bunse C., Kamsties E., Laitenberger O., Laqua R., Muthig D., Paech B., Wust J, Zettel J., *Component-based Product Line Engineering with UML,* Addison Wesley, 2001.

[8] Weiss D. M., and Lai C.T.R., *Software Product Line Engineering:* A Family Based Software Engineering Process, Addison Wesley, 1999.

[9] Lee S., Yang Y., Cho E., Kim S. Rhew S., "COMO: A UML-based Component Development Methodology", APSEC, pp. 54-61, 1999.

[10] Bachmann F., Bass Len, "Managing Variability in Software Architecture," at URL:http://www.sei.cmu.edu/plp/variability.pdf, Software Engineering

Institute(SEI), 2001.
[11] Kang K. C., Cohen, S. G., Novak, W. E. and Peterson, A. S., "Feature-oriented Domain Analysis(FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute(SEI)," November 1990.
[12] Coplien J., Hoffman D., Weiss D., "Commonality and Variability in Software Engineering", IEEE Software, pp. 37-45, November 1998.
[13] Object Management Group, OMG Unified Modeling Language Specification, OMG, September 2001.

소 동 섭
2001년 숭실대학교 컴퓨터학과 공학사. 2001년 3월~현재 숭실대학교 컴퓨터학과 석사과정 재학중. 관심분야: 컴포넌트 개발 방법론, 객체지향 개발 방법론, 소프트웨어 품질 인증, 도메인 공학

신 석 규
1998년 충남대학교 대학원 컴퓨터과학과. 1979년 10월~1981년 5월 KIST(한국과학기술연구원) 전산개발센터 연구원. 1981년 5월~1985년 8월 SANCST(사우디아라비아국립과학기술연구소) 파견(기술고문). 1985년 8월~1995년 6월 KAIST(한국과학기술원) 시스템공학연구소 선임연구원. 1986년 1월~1988년 12월 '88서울올림픽 전산담당관. 1992년 1월~1993년 12월 '93 대전엑스포 회장 운영시스템 개발단장. 1995년 7월~1998년 5월 시스템공학연구소 경영정보과장. 관심분야: 컴포넌트기반 소프트웨어 개발 방법론, 소프트웨어 아키텍처, 컴포넌트 정형명세, 컴포넌트 메트릭

김 수 동
정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 1 호 참조