# 소프트웨어 컴포넌트의 Provided, Required와 Customize인터페이스 설계 기법

## (Methods to Design Provided, Required and Customize Interfaces of Software Components)

박 지 영 † 김 수 동 ††
(Ji Young Park) (Soo Dong Kim)

**요 약** 컴포넌트 기반 개발은 재사용 단위의 컴포넌트를 사용한 경제적인 소프트웨어 개발 패러다임으로 학계와 산업계에 폭넓게 사용되고 있다. 잘 정의된 컴포넌트 인터페이스는 컴포넌트 사이의 저결합도와 의존도를 관리하며, 재사용과 확장성 및 유지보수성을 향상시키는 매개체가 된다. 따라서 컴포넌트가 도입된 이후 컴포넌트 인터페이스의 체계적인 개발 프로세스와 설계 지침에 대한 연구와 방법이 요구되어 왔다. 본 논문에서는 소프트웨어 아키텍쳐 계층 및 기능 유형에 따라 컴포넌트 기능을 제공하는 *Provided*, 요구하는 기능을 명세단위로 제공하는 *Required*와 사용자의 요구에 맞게 컴포넌트를 특화하는 *Customize* 인터페이스를 제시하며, 인터페이스 설계를 위한 기준을 마련하여 체계적인 설계 프로세스 및 지침을 고안한다. 먼저 아키텍쳐 계층별 *Provided* 인터페이스를 식별하기 위해 유즈케이스 모델과 클래스 모델 등에서 추출된 오퍼레이션을 클러스터링하며, 컴포넌트 내 식별된 가변성 배치 산출물에 기반하여 *Customize* 인터페이스를 설계한다. 또한 인터페이스 의존도에 따라 컴포넌트 기능 수행시 요구되는 *Required* 인터페이스를 명세로 제공한다. 제시된 설계 지침은 설계 전 과정을 통해 추적성이 보장되며, 사례연구를 통하여 고품질의 컴포넌트 인터페이스를 위한 실용적인 설계 기법의 토대를 마련한다.

**키워드** : 컴포넌트기반개발, 시스템/비즈니스 인터페이스, *Provided/Required/Customize* 인터페이스

***Abstract*** Component-based Development is gaining a wide acceptance as an economical software development paradigm to develop applications by utilizing reusable software components. Well-defined interface manages coupling and cohesion between components, minimizes the effect on the user in case of component evolvement, and enhances reusability, extendibility and maintainability. Therefore, study on systematic development process and design guidelines for component interface has been required since the component has been introduced. In this paper, we propose three types of interfaces based on software architecture layers and functionality types; *Provided* Interface which provides functionality of a component, *Required* Interface which specifies required functionality that is provided by other components, and *Customize* Interface which tailors the component to customer's requirement. In addition, we suggest design criteria for well-designed interface, and systematic process and instructions for designing interface. We firstly cluster operations extracted from use case model and class model to identify *Provided* interfaces, and design *Customize* interfaces based on artifacts for variability. We also specify *Required* interfaces by identifying dependency among interfaces. Proposed interface design method provides traceability, throughout the component interface design. And furthermore, proposed guidelines support practical design for high quality component based on a case study.

**Key words** : Component-Based Development, *System/Business* Interface, *Provided/Required/Customize* Interface

† 비 회 원 : 바산네트워크(주) 연구원
jyup79@naver.com
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@ssu.ac.kr
논문접수 : 2004년 3월 29일
심사완료 : 2004년 8월 19일

## 1. 서 론

컴포넌트 기반 개발(Component-Based Development, CBD)은 재사용 단위의 컴포넌트를 사용하여 소프트웨어의 중복개발을 지양하고 시스템을 보다 쉽게 고도화

하는 경제적인 소프트웨어 개발 패러다임으로 학계와 산업계에 폭넓게 사용되고 있다. 이러한 CBD 방법론의 주요 구성요소인 클래스/객체의 집합단위 컴포넌트는 구현과 인터페이스가 별개의 모듈로 존재하며, 명세화된 인터페이스 문맥에 의존된 단위로 컴포넌트의 기능을 발휘하게 된다. 즉, 컴포넌트는 내부 구현을 캡슐화하고, 정의된 인터페이스를 통하여 환경과 상호작용한다[1,2].

컴포넌트의 핵심 요소인 인터페이스는 *Provided, Required*와 *Customize* 인터페이스로 구성된다. *Provided* 인터페이스는 컴포넌트의 제공되는 기능을 호출하는 인터페이스이며, *Required* 인터페이스는 컴포넌트가 어떠한 기능을 요구하는지를 명세단위로 제공한다. 특별히, *Customize* 인터페이스는 수정없이 컴포넌트가 확장됨을 관리해주는 메커니즘으로 컴포넌트 배치시 소비자의 요구에 맞게 컴포넌트를 특화 할 수 있다.

컴포넌트가 도입된 이래 컴포넌트 인터페이스의 체계적인 개발 프로세스와 설계 지침에 대한 연구와 방법이 요구되어 왔다. 그러나 이에 대한 연구가 거의 이루어지지 않아 개발자들로 하여금 비체계적인 인터페이스 개발 방식을 고수하게 하였으며, 개발된 인터페이스를 사용하는 컴포넌트 역시 재사용과 확장성 및 유지보수성 등의 고유한 장점을 충분히 발휘하지 못하고 있다. 본 논문에서는 소프트웨어 아키텍쳐 계층 및 기능 유형에 따라 컴포넌트의 *Provided, Required*와 *Customize* 인터페이스를 제시한다. 또한 인터페이스 설계 기준을 마련하며, 체계적인 설계 프로세스 및 지침을 고안한다. 제시된 인터페이스 지침은 설계 전 과정을 통해 추적성이 보장되며, 사례연구를 적용하여 인터페이스의 실용적인 설계 기법을 마련한다.

논문의 구성은 다음과 같다. 2장은 관련연구로 인터페이스 설계방법을 제안하는 기존 연구를 소개한다. 3장은 본 논문에서 제시하는 컴포넌트 인터페이스에 대해 설명한다. 4장은 컴포넌트 인터페이스를 위한 설계 프로세스를 제시한다. 5장부터 8장까지는 4장에서 정의한 인터페이스 설계 프로세스의 각 단계별 지침 및 산출물을 제안한다. 9장은 제시된 프로세스를 기반으로 한 사례연구를 제시한다. 10장에서는 본 논문의 평가를 수행하며, 11장에서 결론을 제시한다.

## 2. 관련 연구

Cheesman 기법은 컴포넌트 기반의 소프트웨어를 설계하기 위해 시스템 요구사항부터 명세까지의 프로세스를 제공한다[3]. 또한 어플리케이션 아키텍쳐 계층에 따른 시스템 및 비즈니스 인터페이스 개념을 도입하여, UML 기반의 컴포넌트와 인터페이스 명세화를 시도한다. 제시된 기법에서의 시스템 인터페이스는 유즈케이스와 1:1로 정의되며, 유즈케이스 명세서에 표현된 시스템과 액터(Actor)의 상호작용은 오퍼레이션으로 식별된다. 비즈니스 인터페이스는 클래스간 연관관계를 나타낸 비즈니스 유형모델을 기반으로 한다. Cheesman 기법은 계층별 인터페이스 설계를 제시하였으나 세부적인 지침과 컴포넌트 확장을 위한 *Customize* 인터페이스 설계 방법 등의 보안이 필요하다.

Catalysis 방법론은 컴포넌트의 재사용 및 확장성을 위한 개발 프로세스와 산출물을 제시한다[4]. 이는 비즈니스 설계부터 구현까지의 모든 산출물 추적이 가능하며, 패턴을 적용한 유연한 프로세스를 제공한다. 이 기법에서 제시한 컴포넌트 인터페이스 유형은 컴포넌트 기능을 제공하는 상위 인터페이스와 특화를 위한 하위 인터페이스이다. 상위 인터페이스는 클래스의 public 오퍼레이션으로 구성되며, 하위 인터페이스는 플러그인 메커니즘을 사용한 매개변수화 기법을 적용한다. Catalysis 방법론은 기능 제공 인터페이스와 특화 인터페이스를 제시하였으나, 추가적인 인터페이스 분류와 세부 설계방법이 요구된다.

Katharine 기법은 컴포넌트 재사용성을 위한 설계 원리와 방법을 제시한다[5]. 이는 아키텍쳐 계층별로 컴포넌트 유형을 정의하였으며, 인터페이스에 대한 정의 및 특화 인터페이스를 통한 가변성 설정 지침을 제공한다. 특화 인터페이스는 가변성을 위한 블랙박스 컴포넌트의 인터페이스이며, 가변성이 설정되는 가변점에 따라 위임, 매개변수화, 상속 등의 특화 방법을 제시한다. Katharine 기법은 컴포넌트 및 인터페이스 식별과 가변성 설정 방법에 대한 분류가 세부적이나 각 인터페이스 설계를 위한 상세한 지침이 요구되며, 가변성을 관리하는 인터페이스의 명세 방법이 필요하다.

## 3. 컴포넌트 인터페이스

### 3.1 아키텍쳐 계층

소프트웨어 아키텍쳐는 유연한 어플리케이션을 위해 각 계층의 목적에 맞는 소프트웨어 집합으로 구성된다[6]. 이는 인터페이스를 통해 계층간 상호작용을 한다. 본 논문에서는 그림 1과 같이 사용자 부분의 프리젠테이션 계층과 서버 부분의 시스템 서비스 계층 및 비즈니스 객체 계층으로 아키텍쳐를 구별한다. 또한 계층에 따라 사용자 인터페이스, 시스템 인터페이스와 비즈니스 인터페이스로 구분하나, 본 논문에서는 서버 부분의 시스템 및 비즈니스 인터페이스를 중점하여 다룬다.

시스템 인터페이스는 시스템 전체 기능을 통제하는 시스템 서비스 계층의 인터페이스이다. 즉, 프리젠테이션 계층에서 요청받은 기능을 수행하기 위해 데이타와 관련된 비즈니스 객체 계층과 연동하는 어플리케이션의

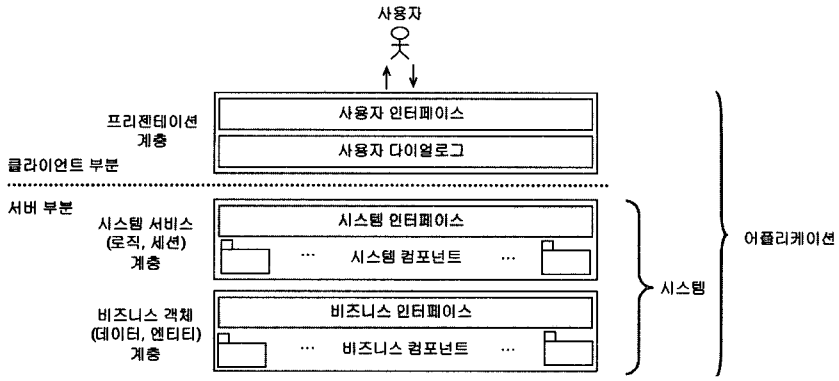



그림 1 소프트웨어 아키텍처 계층

핵심 인터페이스이다.

비즈니스 인터페이스는 재사용 가능한 비즈니스 객체 계층의 인터페이스로, 시스템 인터페이스 호출에 의해 데이타 접근 및 저장 등의 기능을 하는 비즈니스 컴포넌트로 흐름을 수행한다. 단, 비즈니스 인터페이스는 상위 계층의 시스템 인터페이스를 호출할 수 없다[6].

### 3.2 인터페이스 유형

본 논문에서는 그림 2와 같이 컴포넌트 인터페이스를 *Provided*, *Required*와 *Customize* 인터페이스의 세 가지 유형으로 정의한다.

*Provided* 인터페이스는 시스템 및 비즈니스 컴포넌트가 제공하는 기능군을 정의한다. 이는 컴포넌트의 주요 기능을 결정하는 인터페이스로, 일반적인 컴포넌트 인터페이스를 지칭한다.

*Required* 인터페이스는 컴포넌트 실행시 요구하는 다른 컴포넌트의 기능군을 명세한다. 물리적인 인터페이스를 할당하지 않고, 명세서를 제공하여 컴포넌트간 의존 관계를 명시한다.

*Customize* 인터페이스는 컴포넌트 내 가변성을 설정하여, 컴포넌트를 수정하지 않고 확장성을 제공하는 기능군이다. 이는 소비자 중심의 기능 특화를 목적으로 한다.

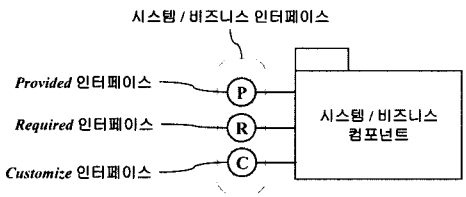



그림 2 인터페이스의 세 가지 유형

### 3.3 설계 기준

컴포넌트 인터페이스는 다양한 기준에 의해 설계됨으로써 시스템 성능을 보장 할 수 있다. 본 논문은 재사용과 확장성 및 유지보수성를 위한 컴포넌트 인터페이스 설계 기준을 제시한다.

**기능 고유성:** 유사한 인터페이스 기능의 중복적인 소개는 최소화한다[7]. 즉, 요구사항 명세서에서 분석된 기능성 중 유사한 기능성이 여러 인터페이스에서 중복적으로 소개되는 것은 경제적이지 못하므로 이를 지양해야 한다.

**기능 고응집성:** 단일 인터페이스에서 제공될 수 있는 기능의 분산은 최소화한다. 즉, 하나의 인터페이스로 설계될 수 있는 기능성을 여러 인터페이스로 분산하지 않고, 단일 인터페이스에서 제공하여 기능 응집도를 보전해야 한다.

**기능 저결합성:** 하나의 인터페이스는 독립적인 기능성을 수행하여 다른 인터페이스의 기능 변경에도 무결성을 보장해야 한다. 이는 인터페이스간 낮은 결합 관계를 유지하게 하여 컴포넌트의 확장성 및 높은 유지보수성을 가능하게 한다.

**기능 특화성:** 컴포넌트는 개발과정에서 식별된 공통성 제공시 소비자에 특화된 가변성이 유도될 수 있다. 이때, 인터페이스는 컴포넌트 내부 요소를 변경하지 않고도 변경가능한 부분을 설정하는 기능을 제공하도록 한다.

**기능 명세성:** 인터페이스 설계시 일정 규칙에 의한 문맥 및 기능 명세는 안정된 인터페이스의 기능성을 관리하고 인터페이스간 상호작용을 원활하게 한다. 이때, 인터페이스는 고유한 이름과 사전 및 사후 조건 등을 인터페이스 명세서에 명시하고, 제시된 문맥에 맞게 기능성을 관리할 수 있어야 한다.

## 4. 인터페이스 설계 프로세스

본 논문에서는 앞서 소개한 컴포넌트 인터페이스 아키텍처 계층과 기능 유형에 따라 4단계의 인터페이스 설계 프로세스를 제시하며, 단계마다 세부적인 지침과 활동을 제공한다. 그림 3은 컴포넌트 *Provided*, *Required*와

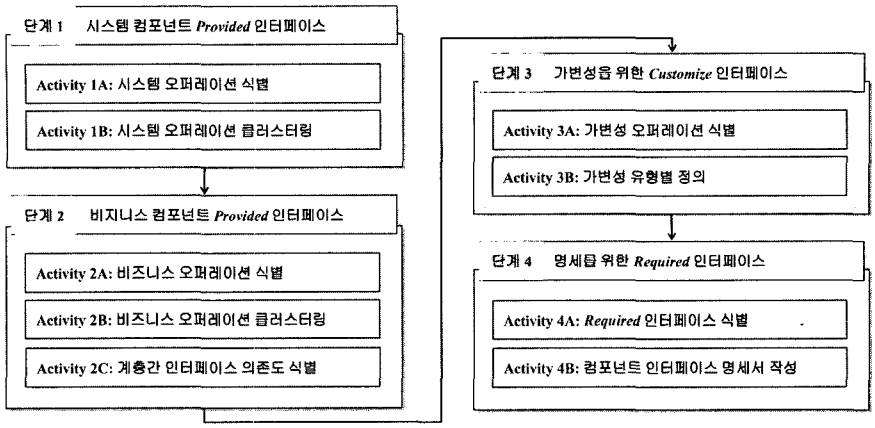



그림 3 컴포넌트 인터페이스 설계 프로세스

*Customize* 인터페이스를 위한 설계 프로세스이다.

단계 1에서는 시스템 서비스 계층에서 기능을 제공하는 시스템 *Provided* 인터페이스(이하 시스템 인터페이스)를 설계한다. 시스템 인터페이스는 사용자가 원하는 기능성을 나타낸 유즈케이스 모델과 사용자와 시스템간 상호작용을 나타낸 시스템 순차도(System Sequence Diagram)[8]를 근거로 시스템 오퍼레이션을 식별한 후, 이를 일정 기준에 따라 클러스터링(Clustering)하여 설계한다.

단계 2에서는 비즈니스 객체 계층에서 제공하는 비즈니스 *Provided* 인터페이스(이하 비즈니스 인터페이스)를 설계한다. 비즈니스 인터페이스는 비즈니스의 순차적인 로직을 나타내는 순차도(Sequence Diagram)[8]에 참여하는 클래스를 식별하고, 식별된 클래스를 일정기준에 따라 클러스터링하여 설계한다. 또한, 단계 1에서 설계된 시스템 인터페이스와 비즈니스 인터페이스의 기능적 요구에 따라 계층간 의존도를 파악한다.

단계 3에서는 단계 1과 2에서 설계된 *Provided* 기능성 중 기능 특화를 위한 *Customize* 인터페이스를 설계한다. *Customize* 인터페이스는 요구사항 분석 산출물 중 컴포넌트 가변성을 식별하여 정리한 '가변성 배치표'[9]를 바탕으로 가변성 오퍼레이션을 식별한 후, 가변성 유형에 따라 설계된다.

단계 4에서는 단계 2에서 파악된 인터페이스간 의존도에 따라 인터페이스 수행시 다른 인터페이스를 요구하는 *Required* 인터페이스를 설계한다. 이는 인터페이스 명세서에서 제공된다.

그림 4는 각 Activity별 산출물의 추적성을 표현하였다. 즉, 단계별로 입력물과 산출물을 통해 인터페이스를 식별, 설계하며 최종적으로 인터페이스 명세서를 산출하게 된다. 이때, 유즈케이스 모델, 클래스 모델, 시스템 순차도, 순차도, 가변성 배치 표와 인터페이스 명세서 외의 산출물은 본 논문에서 제시한 고유한 산출물로, 이에 대한 정의 및 정보는 각 Activity에서 확인할 수 있다.

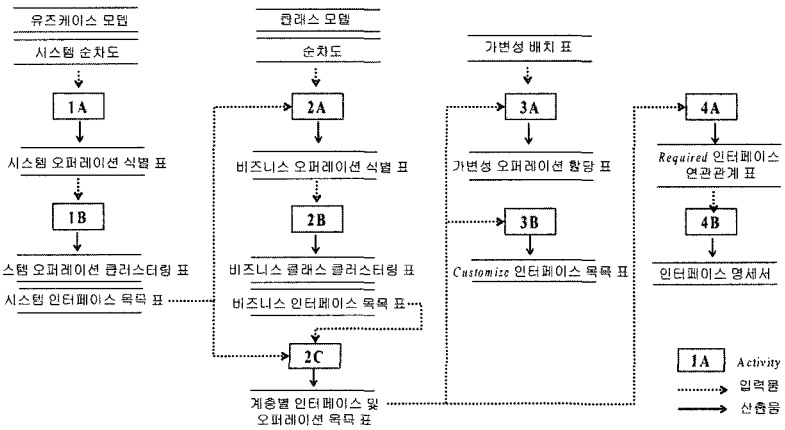



그림 4 프로세스 단계별 입력물 및 산출물

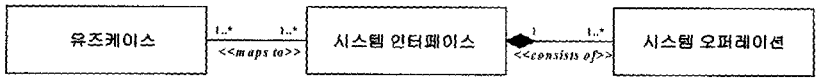



그림 5 유즈케이스, 시스템 인터페이스와 시스템 오퍼레이션 연관관계

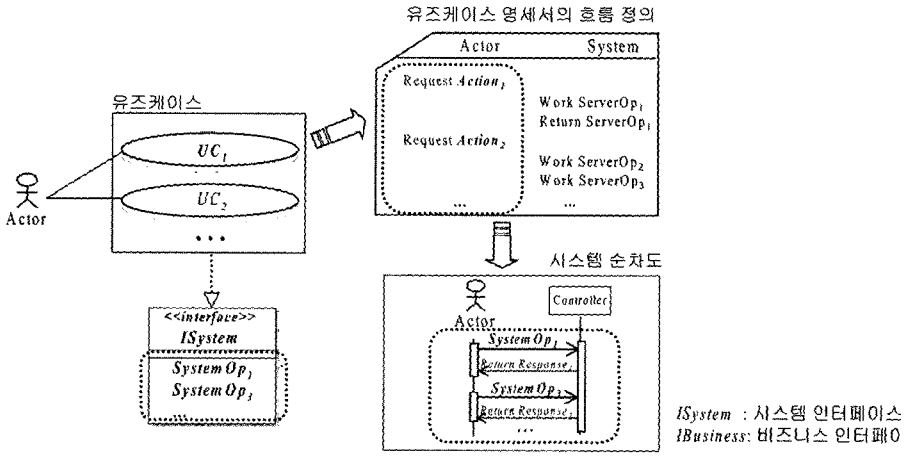



그림 6 시스템 오퍼레이션 식별을 위한 유즈케이스 확장 모델

## 5. 시스템 컴포넌트 Provided 인터페이스 설계

단계 1은 어플리케이션 범위내 존재하는 시스템 또는 비즈니스 컴포넌트를 중재하며, 사용자 요구 기능의 매개체가 되는 시스템 인터페이스를 설계한다. 시스템 인터페이스를 구성하는 오퍼레이션은 유즈케이스 모델과 시스템의 구체적인 로직은 고려되지 않은 시스템 순차도를 토대로 식별된다[6]. 그림 5는 하나 이상의 유즈케이스에서 발견되는 시스템 인터페이스와 시스템 오퍼레이션간 연관관계를 나타낸다.

### 5.1 Activity 1A: 시스템 오퍼레이션 식별

Activity 1A는 유즈케이스 모델을 사용하여, 시스템 인터페이스를 구성하는 시스템 오퍼레이션을 식별한다. 시스템 오퍼레이션은 다음과 같은 Step을 적용하여 식별할 수 있다.

**Step 1. 유즈케이스 명세서를 통해 사용자와 시스템간 상호작용을 파악한다.**

**Step 2. 유즈케이스 명세서의 흐름 수행을 위해 요구되는 기능성을 동적으로 표현한 시스템 순차도에서 시스템 오퍼레이션을 식별한다.**

그림 6은 유즈케이스와 유즈케이스 명세서 및 이를 동적으로 나타낸 시스템 순차도의 확장관계를 나타낸다. 이때, 시스템 순차도에서 수행되는 한 개 이상의 오퍼레이션은 시스템 인터페이스를 구성하기 위한 오퍼레이션으로 식별될 수 있다.

**Step 3. 식별된 시스템 오퍼레이션은 '시스템 오퍼레이션 식별 표'를 통해 정리된다.**

각 유즈케이스에서 식별된 시스템 오퍼레이션은 표 1을 통해 정리된다. 이때, 유즈케이스와 연관되는 시스템 오퍼레이션에 일련 번호를 부여하여 다른 유즈케이스의 시스템 오퍼레이션과 구별한다. 단, 여러 유즈케이스에서 공통으로 식별된 시스템 오퍼레이션은 동일한 일련번호를 부여받는다.

표 1 시스템 오퍼레이션 식별 표

| 유즈케이스 | | 시스템 오퍼레이션 | |
|---|---|---|---|
| 번호 | 이름 | 번호 | 이름 |
| 01 | $UC_1$ | 01 | $SystemOp_1()$ |
| | | 03 | $SystemOp_3()$ |
| | | 05 | $SystemOp_5()$ |

### 5.1 Activity 1B: 시스템 오퍼레이션 클러스터링

Activity 1B는 식별된 시스템 오퍼레이션을 시스템 인터페이스에 할당하기 위한 시스템 오퍼레이션 클러스터링 기준을 제시한다. 또한, 명명 규칙에 따라 시스템 인터페이스를 정의한다. 시스템 인터페이스는 다음과 같은 Step을 적용하여 설계할 수 있다.

**Step 1. 시스템 오퍼레이션 클러스터링 기준의 할당 변수값을 지정한다.**

Step 1에서는 시스템 인터페이스에 참여하는 시스템 오퍼레이션을 클러스터링하기 위한 5가지 기준을 제시한다. 제시된 클러스터링 기준은 인터페이스에 할당하고자 하는 시스템 오퍼레이션 $SystemOp_i$와 $SystemOp_j$간 기능적 의존도를 효과적으로 측정할 수 있게 한다[10].

또한 의존도가 파악되면 적용하고자 하는 프로젝트 특징에 따라 최소 0에서 최대 1까지의 가중치를 부여하여, 능동적이며 확장성 있는 클러스터링이 될 수 있다.

(기준 1) 공통 기능 범주(Category)에 속할 경우, 변수값 $c_1$ 할당

소프트웨어 요구사항 명세서의 전체 범주 중 공통 기능내에 속하는 시스템 오퍼레이션은 다른 범주에 속하는 오퍼레이션보다 밀접한 연관관계에 있다. 만일, $SystemOp_i$와 $SystemOp_j$가 동일한 범주에 속하면 변수값 $c_1$에 1을 할당하며, 속하지 않으면 변수값 $c_1$에 0을 할당한다.

(기준 2) 동일 액터에 의해 초기화(Initiate)되는 유즈케이스에 속할 경우, 변수값 $c_2$ 할당

동일 액터로부터 초기화되는 유즈케이스의 시스템 순차도에 참여하는 시스템 오퍼레이션은 다른 유즈케이스의 오퍼레이션보다 밀접한 연관관계에 있다. 이때, 액터가 초기화하는 유즈케이스의 시스템 오퍼레이션에 가중치를 할당한다. 단, 유즈케이스는 하나 이상의 액터에 의해 초기화 될 수 있으며, 측정 변수값은 다음과 같이 정의될 수 있다.

$$c_2 = \frac{|actors\ initiating\ UC_i \cap actors\ initiating\ UC_j|}{|actors\ initiating\ UC_i \cup actors\ initiating\ UC_j|}$$

즉, $c_2$는 유즈케이스 $Uc_1$와 $Uc_1$를 초기화하는 액터의 비율로 측정되며, $0<c_2<1$의 값을 갖는다. $c_2$의 값이 1에 가까울수록 동일 액터가 공유하는 유즈케이스의 오퍼레이션이 된다.

(기준 3) 반드시 포함(Inclusion)되는 시스템 오퍼레이션이 정의된 유즈케이스에 속할 경우, 변수값 $c_3$ 할당

유즈케이스 수행시 반드시 포함해야하는 시스템 오퍼레이션이 존재할 수 있다. 이때, 포함조건의 오퍼레이션을 공유하는 오퍼레이션은 다른 오퍼레이션보다 밀접한 연관관계에 있으며, 다음과 같이 측정 변수값이 정의될 수 있다.

$$c_3 = \frac{|SystemOps\ manipulated\ by\ UC_i \cap SystemOps\ manipulated\ by\ UC_j|}{|SystemOps\ manipulated\ by\ UC_i \cup SystemOps\ manipulated\ by\ UC_j|}$$

이러한 포함조건의 시스템 오퍼레이션은 다수 유즈케이스의 시스템 오퍼레이션이 공유할 수 있으나, 인터페이스로 추출된 이후에는 다른 인터페이스와 중복을 삼가하여 유사한 기능을 수행하는 인터페이스의 획득을 최소화해야 한다[7]. $c_3$의 변수값은 $c_2$와 동일하게 적용된다.

(기준 4) 동일 기능을 위해 시스템 순차도에서 호출되는 경우, 변수값 $c_4$ 할당

유즈케이스 실행시 동일 기능 제공을 위해 참여하는 시스템 오퍼레이션은 오퍼레이션간에 응집성이 강하므로 다른 오퍼레이션보다 밀접한 연관관계에 있다. $SystemOp_i$와 $SystemOp_j$가 동일 기능 제공에 참여하면 변수값 $c_4$에 1을 할당하며, 참여하지 않으면 0을 할당한다.

(기준 5) 도메인 전문가에 의한 높은 우선순위(Priority)로 평가된 경우, 변수값 $c_5$ 할당

고려된 기준의 합산점수가 동일하거나 기준의 애매성이 존재할 경우 전문가의 경험에 따라 변수값 $c_5$에 1값을 할당하여 가산점을 부여할 수 있다.

**Step 2. 설정된 변수값의 합계에 따라 시스템 오퍼레이션 클러스터링을 수행한다.**

표 2는 Step 1에서 제시한 5가지 기준으로 시스템 인터페이스를 획득하기 위해 시스템 오퍼레이션을 클러스터링한다.

$\Sigma(=\Sigma c_i * w_i)$는 시스템 오퍼레이션마다 제시한 기준의 변수값에 프로젝트에 따라 다르게 주어지는 가중치를 적용한 합산점수이다. 이때, $\Sigma$를 평가하기 위한 적정점수 $S$를 선정하며 $\Sigma$가 S 이상의 값을 만족해야 시스템 인터페이스의 오퍼레이션으로 추출되게 한다. 이때, $S$가 낮을수록 시스템 오퍼레이션으로 이루어지는 시스템 인터페이스 기능 응집력이 낮아지며, 높을수록 시스템 인터페이스 기능이 산재할 수 있다.

**Step 3. 클러스터링 결과를 바탕으로 시스템 인터페이스 명명 규칙을 정의한다.**

Step 1과 2에서 획득된 시스템 인터페이스는 그림 7과 같이 메타언어 BNF 형식을 활용하여 정의한다. 이때, (4)와 같이 인터페이스 이름에 접두사 "*IS*"를 두어

표 2 시스템 오퍼레이션 클러스터링 표

| 시스템 오퍼레이션 번호 | 기 준 | 시스템 오퍼레이션 번호 | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 01 | | 02 | | 03 | | … |
| | | $c_i$ | $c_i*w_i$ | $c_i$ | $c_i*w_i$ | $c_i$ | $c_i*w_i$ | … |
| 01 | 범주 | $c_1$ | | $c_1$ | | $c_1$ | | |
| | 액터 | $c_2$ | | $c_2$ | | $c_2$ | | |
| | 포함 | $c_3$ | | $c_3$ | | $c_3$ | | |
| | 호출 | $c_4$ | | $c_4$ | | $c_4$ | | |
| | 우선순위 | $c_5$ | | $c_5$ | | $c_5$ | | |
| | 합계 | | $\Sigma c_i * w_i$ | | $\Sigma c_i * w_i$ | | $\Sigma c_i * w_i$ | … |

Interface ::= <Interface Type>; ..... (1)
Interface Type ::= <System Interface Type> | <Business Interface Type>; ..... (2)
System Interface Type ::= <System Provided Interface> | <System Required Interface> | <System Customize Interface>; ... (3)
System Provided Interface ::= "*IS*"<Interface Name><Method Declaration>{[<Semantics>]}; ..... (4)
Method Declaration ::= <Visibility><Return Type><Method Name>(<Parameter List>); ..... (5)
Semantics ::= {[<Pre-condition>]}{[<Post-condition>]}{[<Exception>]}; ..... (6)

그림 7 시스템 인터페이스 정의

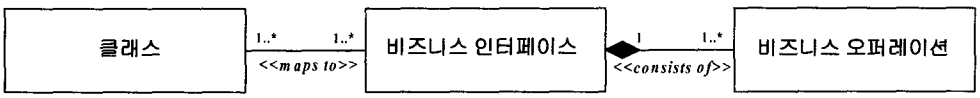



그림 8 클래스, 비즈니스 인터페이스와 비즈니스 오퍼레이션 연관관계

다른 계층의 인터페이스와 구분한다.

**Step 4. 시스템 인터페이스와 연관된 시스템 오퍼레이션을 나열하며, 인터페이스의 기능을 기술한 '시스템 인터페이스 목록 표'를 작성한다.**

표 3은 획득된 시스템 인터페이스 및 시스템 오퍼레이션과 기능을 정리한다.

표 3 시스템 인터페이스 목록 표

| 시스템 인터페이스 | 시스템 오퍼레이션 | | 기능성 |
|---|---|---|---|
| | 번호 | 이름 | |
| *IS*<Interface name$_1$>() | 01<br>03<br>05 | *SystemOp*$_1$()<br>*SystemOp*$_3$()<br>*SystemOp*$_5$() | ... |

## 6. 비즈니스 컴포넌트 Provided 인터페이스 설계

단계 2는 저장소 정보의 접근, 변경을 위해 필요한 비즈니스 인터페이스를 설계한다. 비즈니스 인터페이스는 시스템 인터페이스에 해당하는 순차도 내 메시지 흐름과 참여하는 클래스 및 클래스간 상호작용을 토대로 식별된다. 그림 8은 하나 이상의 클래스에 참여하는 비즈니스 인터페이스와 비즈니스 오퍼레이션간 연관관계를 나타낸다.

### 6.1 Activity 2A: 비즈니스 오퍼레이션 식별

Activity 2A는 표 3의 시스템 오퍼레이션 순차도의 클래스 정보를 이용하여 비즈니스 오퍼레이션을 식별한다. 비즈니스 오퍼레이션은 다음과 같은 Step을 적용하여 식별할 수 있다.

**Step 1. 시스템 오퍼레이션 식별시 사용한 시스템 순차도를 시스템 내부 로직까지 나타내는 순차도로 확장하여 오퍼레이션 흐름을 파악한다.**

**Step 2. 순차도의 오퍼레이션을 수행하기 위해 참여하는 클래스를 식별한다.**

그림 9는 비즈니스 로직을 표현한 순차도 및 로직에 참여하는 클래스 식별을 통해 비즈니스 인터페이스를 추출하는 관계를 나타낸다. 순차도에 나타난 오퍼레이션은 클래스의 오퍼레이션이며, 다음 단계의 클래스 클러스터링에 따라 비즈니스 인터페이스를 구성하는 비즈니스 오퍼레이션으로 식별된다.

**Step 3. 시스템 오퍼레이션의 순차도에 참여하는 비즈니스 오퍼레이션과 식별된 클래스를 기술한 '비즈니스 오퍼레이션 식별 표'를 작성한다.**

각 순차도에서 식별된 클래스는 표 4를 통해 정리되며, 여러 순차도에 참여하는 하나의 클래스는 동일한 일련번호를 부여받는다.

표 4 비즈니스 오퍼레이션 식별

| 순차도 | | 클래스 | |
|---|---|---|---|
| 시스템 오퍼레이션 | 비즈니스 오퍼레이션 | 번호 | 이름 |
| *SystemOp*$_1$() | *BusinessOp*$_1$()<br>*BusinessOp*$_5$() | 01<br>02 | *Class A*<br>*Class C* |
| *SystemOp*$_3$() | *BusinessOp*$_3$() | 03 | *Class D* |

### 6.2 Activity 2B: 비즈니스 오퍼레이션 클러스터링

Activity 2B는 Activity 2A에서 식별된 클래스의 클러스터링 기준 및 클래스 정보관리를 위한 핵심 클래스 선정 기준을 제시하여 비즈니스 인터페이스를 식별한다. 클래스 클러스터링 기법은 클래스를 식별하여 후보 컴포넌트로 설계하는 CBD96이나 유즈케이스와 클래스간 관계에 따라 우선 순위를 두고 클러스터링했던 COMO 방법론에서도 찾아볼 수 있다[11,12]. 그러나 기존 컴포넌트 식별 연구와 달리 본 논문의 클러스터링 기법은 컴포넌트 인터페이스를 위한 클러스터링에 초점을 맞춘 것으로, 이전 단계에서 추출된 시스템 인터페이스의 순차도에 참여하는 클래스를 먼저 고려한다. 또한 시스템 계층과 연동되어 기능을 수행하는 클러스터링된 클래스 군을 제어하는 핵심 클래스를 추출하여 이를 비즈니스 인터페이스로 식별한다. 이를 위해 다음과 같은 Step을 적용할 수 있다.

**Step 1. 클래스 클러스터링 기준의 할당 변수값을 지정한다.**

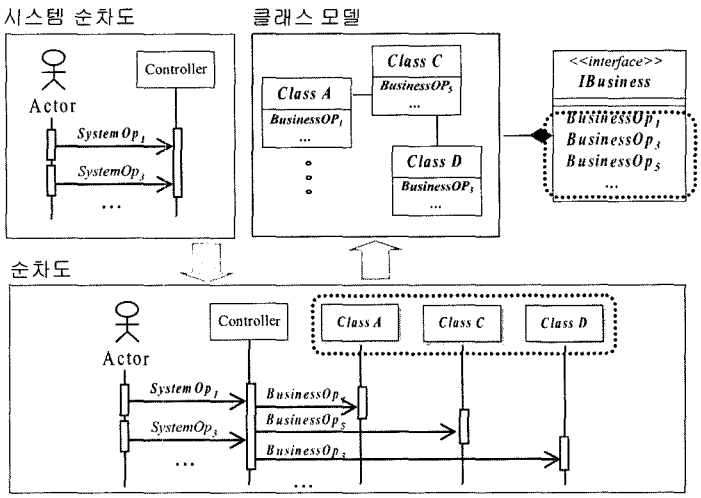



그림 9 비즈니스 오퍼레이션 식별을 위한 순차도 확장 모델

Step 1에서는 클래스 $Class_i$와 $Class_j$의 기능간 의존도를 평가하여 비즈니스 오퍼레이션을 위한 클래스 클러스터링의 5가지 기준을 제시한다. 클래스간 의존도가 파악되면 시스템 인터페이스와 동일하게 가중치를 부여할 수 있다.

(기준 1) 동일 시스템 인터페이스의 순차도에 참여할 경우, 변수값 $c_1$ 할당

동일 시스템 인터페이스의 순차도에 참여하는 클래스의 오퍼레이션은 시스템 인터페이스 수행시 호출되는 비즈니스 인터페이스의 커다란 범주가 되며, 다음과 같은 측정 변수값이 정의될 수 있다.

$$c_1 = \frac{|classes\ of\ manipulated\ by\ ISystem_i \cap classes\ of\ manipulated\ by\ ISystem_j|}{|classes\ of\ manipulated\ by\ ISystem_i \cup classes\ of\ manipulated\ by\ ISystem_j|}$$

즉, $c_1$은 $Class_i$와 $Class_j$가 시스템 인터페이스의 순차도에서 참여하는 비율로 측정되며, $0<c_1<1$의 값을 할당받게 된다. $c_1$의 값이 1에 가까울수록 동일 시스템 인터페이스의 순차도에 참여하는 클래스가 되며, 클래스의 오퍼레이션은 서로 연관관계를 갖게 된다.

표 5 비즈니스 클래스 클러스터링 표

| 클래스 번호 | 기 준 | 시스템 오퍼레이션 번호 | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 01 | | 02 | | 03 | | … |
| | | $c_i$ | $c_i*w_i$ | $c_i$ | $c_i*w_i$ | $c_i$ | $c_i*w_i$ | … |
| 01 | 시스템 인터페이스 참여 | $c_1$ | | $c_1$ | | $c_1$ | | |
| | 시스템 오퍼레이션 참여 | $c_2$ | | $c_2$ | | $c_2$ | | |
| | 유즈케이스 역할 | $c_3$ | | $c_3$ | | $c_3$ | | |
| | 클래스간 연관관계 | $c_4$ | | $c_4$ | | $c_4$ | | |
| | 우선순위 | $c_5$ | | $c_5$ | | $c_5$ | | |
| | 합계 | | $\sum c_i * w_i$ | | $\sum c_i * w_i$ | | $\sum c_i * w_i$ | … |

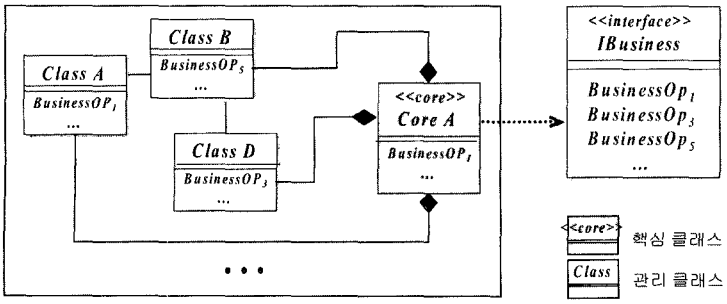



그림 10 핵심 클래스 책임 할당 모델

Business Interface Type ::= <Business Provided Interface> | <Business Required Interface> | <BusinessCustomize Interface>;
Business Provided Interface ::= "*IB*"<Interface Name><Method Declaration>{[<Semantics>]}; ... ... (4)

그림 11 비즈니스 인터페이스 정의

표 6 비즈니스 인터페이스 목록 표

| 비즈니스인터페이스 | 클래스 및 오퍼레이션 | | | | | 기능성 |
|---|---|---|---|---|---|---|
| | 핵심클래스 | 핵심클래스오퍼레이션 | 관리클래스 | 관리클래스오퍼레이션 | 오퍼레이션탈락 | |
| *IB*<Interface name$_1$>() | *Core A* | *BusinessOp*$_1$()<br>*BusinessOp*$_3$()<br>*BusinessOp*$_5$() | *Class A* | *BusinessOp*$_1$()<br>*BusinessOp*$_9$() | *BusinessOp*$_9$()<br>*BusinessOp*$_{10}$()<br>: 시스템 자체처리 | ... |
| | | | *Class B* | *BusinessOp*$_5$() | | |
| | | | *Class D* | *BusinessOp*$_3$()<br>*BusinessOp*$_{10}$() | | |

표 7 계층별 인터페이스 및 오퍼레이션 목록 표

| 인터페이스 | | 오퍼레이션 | | 수행기능 |
|---|---|---|---|---|
| 시스템 계층 | 비즈니스 계층 | 시스템 계층 | 비즈니스 계층 | |
| *IS*<Interface name$_1$>() | *IB*<Interface name$_1$>() | *SystemOp*$_1$() | *BusinessOp*$_1$()<br>*BusinessOp*$_5$() | ... |
| | | *SystemOp*$_3$() | *BusinessOp*$_3$() | |
| | *IB*<Interface name$_2$>() | *SystemOp*$_5$() | *BusinessOp*$_7$() | ... |

(기준 2) 동일 시스템 오퍼레이션의 순차도에 참여할 경우, 변수값 $c_2$ 할당

하나의 시스템 인터페이스를 위해 여러 시스템 오퍼레이션이 실행되며, 이중 동일 시스템 오퍼레이션의 순차도에 참여하는 클래스는 서로 밀접한 연관관계에 있다. 이는 시스템 인터페이스에 속한 여러 시스템 오퍼레이션 실행에 참여하는 클래스를 클러스터링 하는 (기준 1)보다 세분화된 범위이며, 다음과 같은 측정 변수값이 정의될 수 있다. $c_2$의 변수값은 $c_1$과 동일하게 적용된다.

$$c_2 = \frac{|classes\ of\ manipulated\ by\ SystemOp_i \cap classes\ of\ manipulated\ by\ ISystemOp_j|}{|classes\ of\ manipulated\ by\ ISystemOp_i \cup classes\ of\ manipulated\ by\ ISystemOp_j|}$$

(기준 3) 유즈케이스 역할(Role)에 의해 정의된 경우, 변수값 $c_3$ 할당

비즈니스 인터페이스는 데이타를 관리하는 인터페이스로 생성, 조회, 수정, 삭제의 역할중 유사 기능을 수행하는 유즈케이스내 클래스들은 서로 밀접한 연관관계에 있다. 생성과 삭제는 데이타 수명에 관계하며, 수정은 데이타 상태 변경, 조회는 데이타 검색 역할을 하게 된다. 각 역할에 따라 0<Weight(Search)<Weight(Update)<Weight(Create),Weight(Delete)<1의 가중치를 0.3, 0.7과 1.0로 부여하여 다음과 같은 측정 변수값이 정의될 수 있다.

$c_3$ = *weight of role of classes in use cases*

(기준 4) 클래스간 연관관계(Relationship)에 의해 정의된 경우, 변수값 $c_4$ 할당

클래스는 클래스간 연관(Association), 집합(Aggregation) 또는 복합(Composition)의 관계(Relationship)[8]를 맺을 수 있다. 이러한 관계를 통해 클래스 클러스터링을 수행함으로써 비즈니스 인터페이스 기능의 독립성이 높아지고 인터페이스간 결합력이 낮아질 수 있다. 먼저, 연관은 데이타간 영구적인 상호작용으로 높은 관계를 맺는다. 집합은 특정 기능 요구에 의한 관계를 맺게 된다. 복합은 집합보다 강하며, 생성과 소멸이 직결되어 있는 관계를 가진다. 각 관계에 따라 0<Aggregation<Association<Comosition<1의 가중치를 0.3, 0.7과 1.0로 부여하여 다음과 같은 측정 변수값이 정의될 수 있다.

$c_4$ = *weight of relationship between classes*

(기준 5) 도메인 전문가에 의한 높은 우선순위로 평가된 경우, 변수값 $c_5$ 할당

기준 1에서 4까지 적용된 클러스터링의 합산점수가 동일하거나 기준의 애매할 경우, 전문가의 경험에 따라 $c_5$에 1값을 할당할 수 있다.

**Step 2. 변수값의 합계에 따라 비즈니스 오퍼레이션을 획득하기 위한 비즈니스 클래스 클러스터링을 수행한다.**

표 5는 앞에서 제시한 5가지 기준으로 클래스를 클러스터링 하는 표를 나타낸다.

$\Sigma$는 제시한 기준에 설정된 변수값과 가중치에 따른 합산점수이다. 이때, 시스템 오퍼레이션 클러스터링과 동일하게 적정 점수 $S$를 선정하여 비즈니스 오퍼레이션의 결합력 또는 응집력을 고려한 클러스터링을 할 수 있다.

**Step 3. 획득된 클러스터링 클래스군의 정보관리를 위한 핵심 클래스를 선정한다.**

표 8 가변성 배치 표

| 가변점 | 가변성 유형 | 가변치 집합 | 초기값 | 가변성 범위 | 설정내용 |
|---|---|---|---|---|---|
| $SVP_1$ | 로직 | {$V_1$, $V_2$} | $V_1$ | 식별 | 고유번호 |
| $BVP_3$ | 로직 | {$V_3$, $V_4$, $V_5$} | $V_3$ | 식별 | … |
| $BVP_1$ | 워크플로우 | { } | 없음 | 미식별 | … |

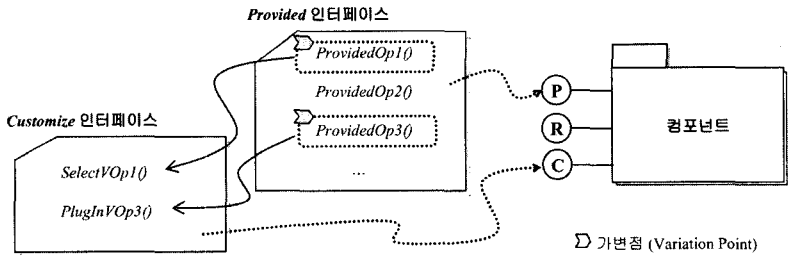



그림 12 가변 기능성을 위한 Customize 인터페이스

표 9 가변성 오퍼레이션 할당 표

| *Customize* 인터페이스범주 | 가변성 오퍼레이션 정보 | | | | | |
|---|---|---|---|---|---|---|
| 기본 인터페이스 | 가변점 | 가변성 유형 | 가변치 집합 | 기본값 | 가변성 범위 | 설정내용 |
| *IS*<Interface name$_1$>() | $SVP_1$ | 논리 | {$V_1$, $V_2$} | $V_1$ | 식별 | 고유번호 |
| *IB*<Interface name$_4$>() | $BVP_1$ | 워크플로우 | { } | 없음 | 미식별 | … |
| | $BVP_3$ | 논리 | {$V_3$, $V_4$, $V_5$} | $V_3$ | 식별 | … |

System Customize Interface ::= *"IC"*<Interface Name><Method Declaration>{[<Semantics>]}; … … (4-1)
Business Customize Interface ::= *"IC"*<Interface Name><Method Declaration>{[<Semantics>]};… … (4-2)

그림 13 Customize 인터페이스 정의

Customize Operation ::=<Variation Scope>;…… (1)
Variation Scope ::= <Close Scope> | <Open Scope>;…… (2)
Close Scope ::= <Close Variation Type>;…… (3)
Close Variation Type ::= <Logic Type> | <Workflow Type>; …… (4)
Logic Type ::= *"Select"*<Variation Point Name>(<Variant Type><Actural Arqument>){[<Semantics>]}; … (5)
Semantics ::= {[<Pre-condition>]}{[<Post-condition>]}{[<Exception>]};…… (6)

그림 14 식별 범위 내 논리 가변성 유형의 오퍼레이션 정의

Step 3은 클러스터링 된 클래스들을 관리하기 위한 핵심 클래스를 선정하여 클래스 정보가 어느 인터페이스에서 관리되는지에 대한 명시를 한다. 핵심 클래스 선정기준은 다음과 같다.

(기준 1) 다른 클래스군과 정보간 의존성이 낮거나, 독립적으로 존재할 수 있는 경우

(기준 2) 클래스 데이타 및 오퍼레이션의 의미가 클래스군을 식별할 수 있는 경우

핵심 클래스의 오퍼레이션은 비즈니스 인터페이스를 구성하는 비즈니스 오퍼레이션이 되며, 이를 통해 관리대상이 되는 클래스(이하 관리 클래스)의 내부 정보를 관리할 수 있게 된다. 관리클래스의 오퍼레이션 중 사용자와 상호작용을 요구하지 않는 오퍼레이션은 핵심 클래스의 오퍼레이션 즉, 비즈니스 오퍼레이션으로 할당되지 않는다. 그림 10은 클러스터링 된 관리 클래스, 핵심 클래스와 비즈니스 인터페이스의 관계를 도식화하였다.

**Step 4. 명명 규칙을 적용하여 비즈니스 인터페이스를 정의한다.**

그림 11은 비즈니스 인터페이스를 BNF형식으로 정의하였다. 비즈니스 인터페이스는 (4)와 같이 인터페이스 이름에 접두사 *"IB"*를 두어 다른 서비스 계층의 인터페이스와 구분한다.

**Step 5. 비즈니스 인터페이스와 연관된 비즈니스 오퍼레이션을 정리한 '비즈니스 인터페이스 목록 표'를 작성한다.**

표 6은 인터페이스로 추출되는 핵심 클래스와 관리클래스 및 각 클래스를 구성하는 오퍼레이션에 대해 비즈니스 인터페이스를 기준으로 정리하였다. 만일 관리클래스 오퍼레이션이 핵심 클래스의 관리를 요구하지 않을 경우, 비즈니스 인터페이스 오퍼레이션으로 추출되지 않는 탈락 사유를 기입할 수 있다.

### 6.3 Activity 2C: 계층간 인터페이스 의존도 식별

Workflow Type ::= *"Select"*<Variation Point Name>(<Variant Type><Actural Arqument>){[<Semantics>]}; ... (5)

그림 15 식별 범위 내 워크플로우 가변성 유형의 오퍼레이션 정의

Open Scope ::= <Open Variation Type>; ... ... (3)
Open Variation Type ::= <Logic Type> | <Workflow Type>; ... ... (4)
Logic Type ::= *"PlugIn"*<Variation Point Name>(<Variant Class Type><Actural object>){[<Semantics>]}; ... (5)

그림 16 미식별 범위 내 논리 가변성 유형의 오퍼레이션 정의

Workflow Type ::= *"PlugIn"*<Variation Point Name>(<Variant Class Type><Actural object>){[<Semantics>]};... (5)

그림 17 미식별 범위 내 워크플로우 가변성 유형의 오퍼레이션 정의

표 10 *Customize* 인터페이스 목록 표

| 범위 | 유형 | 설정매개체 | 접두사 | 매개변수 | 비고 |
|---|---|---|---|---|---|
| 식별 | 논리 | 오퍼레이션 | "Select" | 변수 | *Select*CustomerSearch(int a) |
| | 워크플로우 | 오퍼레이션 | "Select" | 변수 | *Select*CustomerSearch(int b) |
| 미식별 | 논리 | 오퍼레이션 | "PlugIn" | 객체 | *PlugIn*CustomerSearch(CusClass cs) |
| | 워크플로우 | 오퍼레이션 | "PlugIn" | 객체 | *PlugIn*CustomerSearch(CusClass cs) |

Activity 2C는 Activity 1과 Activity 2를 통해 설계된 시스템 및 비즈니스 인터페이스의 계층간 의존도를 식별하여 계층간 상호작용을 파악한다. 표 7은 시스템 및 비즈니스 인터페이스 분류에 따른 오퍼레이션과 계층간 의존도를 정리하였다. 표 7에서 시스템 인터페이스 *IS*<Interface name$_1$>는 기능 수행을 위해 비즈니스 인터페이스 *IB*<Interface name$_1$>와 *IB*<Interface name$_2$>를 호출한다. 이때, *IS*<Interface name$_1$>의 시스템 오퍼레이션 *SystemOp*$_1$은 비즈니스 오퍼레이션 *BusinessOp*$_1$과 *BusinessOp*$_5$를 호출한다.

## 7. 가변성을 위한 Customize 인터페이스 설계

요구사항 명세서의 공통성은 단계 1과 단계 2를 통하여 시스템 및 비즈니스 *Provided* 인터페이스로 설계되었다. 이때, 공통성 중 패밀리 멤버에 특화된 가변성을 요구할 수 있다[9,13]. 기존 연구에서는 컴포넌트 가변성 유형을 오퍼레이션 알고리즘과 로직이 변경되는 논리(Logic) 가변성과 동일한 사후조건을 만족시키나 컴포넌트 내부 메시지 흐름이 변경되는 워크플로우(Workflow) 가변성을 정의하였다. 또한 컴포넌트 내 가변성이 식별된 경우를 식별 범위(Open Scope), 식별되지 않고 컴포넌트 특화 시점에 가변치를 설정 하는 경우를 미식별 범위(Closed Scope)라 한다[14,15].

단계 3에서는 컴포넌트 내 가변성을 식별하는 기존 연구를 바탕으로 하나, 블랙박스 컴포넌트의 식별된 가변성 유형에 따라 가변치를 효과적으로 설정하기 위한 *Customize* 인터페이스 설계 기법 제시를 목적으로 한다.

### 7.1 Activity 3A: 가변성 오퍼레이션 식별

Activity 3A에서는 '가변성 배치'[9] 정보를 바탕으로 *Customize* 인터페이스의 오퍼레이션을 식별하며, 이를 위한 Step은 다음과 같다.

**Step 1. 컴포넌트 개발 요구사항 분석단계 산출물인 '가변성 배치 표'를 수집한다.**

*Customize* 인터페이스의 오퍼레이션 식별을 위해 표 8과 같이 가변점, 가변성 유형 및 가변치 집합, 초기값 등을 분석한 '가변성 배치' 표를 수집한다[9].

**Step 2. '가변성 배치 표'의 가변치를 설정하기 위한 *Customize* 오퍼레이션을 찾는다. 즉, *Provided* 인터페이스 실행시 참조되는 가변성을 정의한다.**

그림 12는 *Provided* 인터페이스 중 가변점에서 가변치 설정을 위한 *Customize* 인터페이스가 유도됨을 나타낸다.

만일, 가변성 오퍼레이션을 *Customize* 인터페이스에 할당할 때 하나의 *Provided* 인터페이스에 속하는 오퍼레이션에서 가변성을 위한 오퍼레이션으로 식별된 경우, 이를 동일한 *Customize* 인터페이스에 할당한다. 이때, 표 7의 인터페이스 및 오퍼레이션 범주를 참조한다.

**Step 3. *Customize* 인터페이스에 할당되는 오퍼레이션을 정리하는 '가변성 오퍼레이션 할당 표'를 작성한다.**

표 8의 가변성 설정을 위한 오퍼레이션 범주는 표 9와 같이 정리할 수 있다.

**Step 4. '가변성 오퍼레이션 할당 표'로부터 *Customize* 인터페이스를 정의한다.**

그림 13은 표 9로부터 추출된 *Customize* 인터페이스를 BNF 형식으로 정의하였다. *Customize* 인터페이스는 (4-1, 2)와 같이 인터페이스 이름에 접두사 'IC'를 붙인다.

### 7.2 Activity 3B: 가변성 유형별 정의

Activity 3B는 가변성 유형에 따라 Activity 3A에서

표 11 *Required* 인터페이스 연관관계

| Required 인터페이스 | | | "Required" 연관관계 |
|---|---|---|---|
| 이름 | 기본 인터페이스 | 요소 인터페이스 | |
| *IR* <Interface name1S>() | *IS*<Interface name1>() | *IB*<Interface name1>() | Call |
| *IR* <Interface name4B>() | *IB*<Interface name4>() | *IB*<Interface name1>() | Set |

**인터페이스 이름 : (기본) "*IS*"<Interface Name>**
**인터페이스 계층 : 시스템 서비스 계층**
**인터페이스 유형 : *Provided***

버 전 : *1.0*
시그네처 : *"IS"*<Visibility><Return Type><Method Name>(<Parameter List>)
사전조건 : *Pre_Condition*
사후조건 : *Post_Condition*
불변조건 : *Constraint_Condition*
설 명 : .....

**인터페이스 이름 : "*IR*"<Interface Name>**
**인터페이스 계층 : 비즈니스 객체 계층**
**인터페이스 유형 : *Required***

기본 인터페이스 : *"IB"*<Interface $Name_1$>
요소 인터페이스 : *"IB"*<Interface $Name_2$>
연 관 관 계 : *Set*
설 명 : .....

**인터페이스 이름 : "*IC*"<Interface Name>**
**인터페이스 계층 : 시스템 서비스 계층**
**인터페이스 유형 : *Customize***

시그네처 : *"Select"*<Variation Point Name>(<Variant Type><Actural Arqument>)
사전조건 : *Pre_Condition*
사후조건 : *Post_Condition*
불변조건 : *Constraint_Condition*

기본 인터페이스 : *"IS"*<Interface $Name_3$>
범 위 : *Open*
가변성 유형 : *Logic*
설정 매개체 : *Operation*
매개변수 : *int a*
설 명 : .....

그림 18 인터페이스 명세서

식별된 가변성 오퍼레이션을 설계한다. 이를 위해 다음과 같은 Step을 적용할 수 있다.

**Step 1. '가변성 오퍼레이션 할당 표'로부터 가변성 유형에 따라 *Customize* 오퍼레이션을 정의한다.**

사용자가 요구한 가변치가 식별되었을 경우, *Customize* 인터페이스의 오퍼레이션은 가변치를 입력받아 영구적으로 설정하는 *Select()* 함수를 사용한다. 그러나 미식별일 경우, 외부로부터 함수나 객체를 가변치로 입력받는 *PlugIn()* 함수를 사용한다.

먼저, 식별 범위 내 논리 가변성 오퍼레이션은 표 9의 가변치 집합을 고려한다. 즉, 사후조건은 동일하지만 실제 설정값에 따라 실행조건이 다르다. 그림 14는 식별 범위 내 논리 가변성을 BNF형식으로 정의한 것으로, (5)와 같이 접두사 *"Select"*와 가변점 이름을 사용하여 오퍼레이션을 정의할 수 있다. 이때, 매개변수는 설정하고자 하는 실제 가변치를 입력받는다.

식별 범위 내 워크플로우 가변성 오퍼레이션은 사용자가 원하는 워크플로우를 특화하는 오퍼레이션으로, 실제 설정값을 매개변수로 입력받아 컴포넌트 내부에 가변적인 워크플로우를 설정한다. 그림 15는 식별 범위내 워크플로우 가변성을 BNF형식으로 정의한 것으로, (5)와 같이 접두사 *"Select"*와 함께 매개변수에 가변 워크플로우를 설정하는 실제 가변치를 입력받는다.

다음으로, 미식별 범위 내 논리 가변성 오퍼레이션은 표 9에서 가변치가 식별되지 않은 것으로 멤버간 요구사항이 다양하거나, 확장 가능성이 존재할 경우 컴포넌트 내부에 가변성 설정내용만 선언한 후 외부에서 해당 클래스 또는 메소드를 Plug-in 한다. 따라서, 그림 16과 같이 접두사 *"PlugIn"*과 가변점 이름을 사용하여 오퍼레이션을 정의하고, 외부에서 제공되는 객체를 매개변수 값으로 입력받는다.

미식별 범위 내 워크플로우 가변성 오퍼레이션은 워크플로우 정보를 제공하는 객체를 외부에서 컴포넌트로 전달해준다. 이는 그림 17의 (5)와 같이 접두사 *"PlugIn"*를 사용하며 워크플로우가 정의된 객체를 매개변수로 입력받는다.

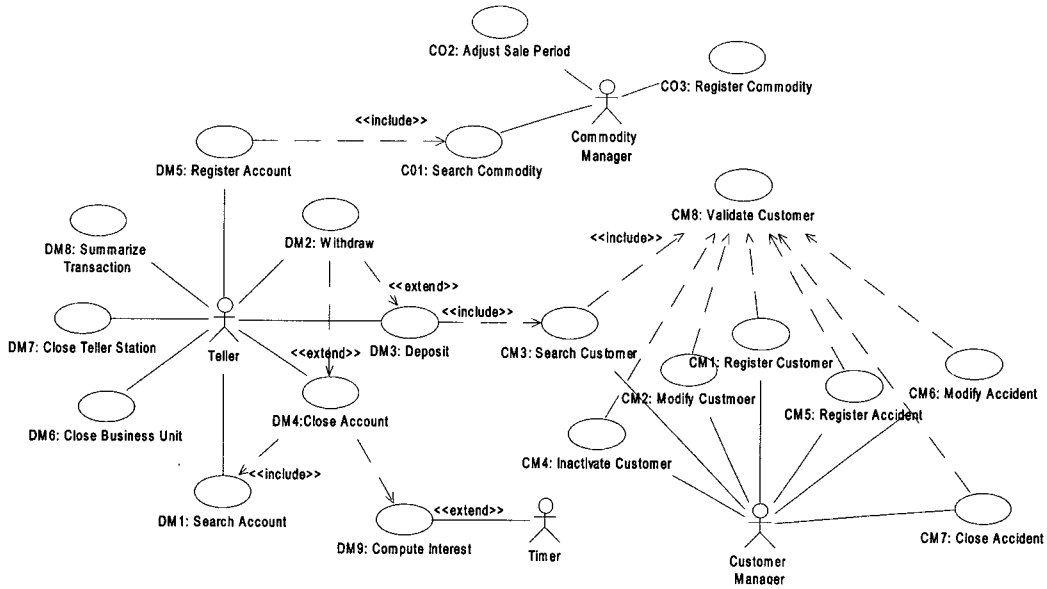



그림 19 은행 도메인의 유즈케이스 다이어그램

표 12 은행 도메인의 시스템 오퍼레이션 식별 표

| 유즈케이스 | | 시스템 오퍼레이션 | |
|---|---|---|---|
| 번호 | 이름 | 번호 | 이름 |
| DM1 | SearchAccount | 01 | searchAccount() |
| DM2 | Withdraw | 010506 | searchAccount()validatePWD()withdraw() |
| DM3 | Deposit | 0107 | searchAccount()deposit() |
| DM4 | CloseAccount | 0308 | searchCustomer()closeAccount() |
| DM5 | RegisterAccount | 0318<br>09 | searchCustomer()<br>searchCommodity()registerAccount() |
| ... | ... | ... | ... |
| CO1 | SearchCommodity | 19 | searchCommodity() |
| CO2 | AdjustSalePeriod | 2021 | adjustCommodityID()modifySaleDays() |
| CO3 | RegisterCommodity | 22 | registerCommodity() |

**Step 2. 정의된 *Customize* 오퍼레이션을 기능범주에 따라 *Customize* 인터페이스로 정제한다.**

표 10은 범위별로 정의한 *Customize* 인터페이스를 정리한다.

## 8. 명세를 위한 Required 인터페이스 설계

지금까지 소프트웨어 아키텍쳐를 기반으로 하여 *Provided* 인터페이스 및 *Customize* 인터페이스를 설계하였다. 단계 4에서는 *Provided* 인터페이스 수행시 요구되는 기능을 외부로부터 제공받기 위한 *Required* 인터페이스를 설계한다. *Required* 인터페이스는 물리적인 형태로 존재하지 않으며, 명세서로 제공된다.

### 8.1 Activity 4A: Required 인터페이스 식별

Activity 4A는 표 7을 바탕으로 *Required* 인터페이스를 식별한다. 이는 기능을 요구하는 인터페이스 연관관계를 계층간이나 계층내로 구분하여 정의하게 된다.

**Step 1. '계층별 인터페이스 및 오퍼레이션 목록표'를 통해 계층간 인터페이스 의존도를 이해한다.**

컴포넌트 인터페이스는 제공하고자 하는 기능을 위해 아키텍쳐의 동일 계층 또는 다른 계층으로부터 컴포넌트 인터페이스를 요구할 수 있다. 표 7에서는 계층별 인터페이스의 의존도를 파악할 수 있다.

**Step 2. 계층간 의존도에 따라 인터페이스 연관관계를 정의한다.**

표 11은 *Required* 인터페이스 이름과 기능을 요구하는 기본 인터페이스 및 기능을 제공해 주는 요소 인터페이스를 명시한다. "Required" 연관관계는 기본 인터페이스와 요소 인터페이스의 연관관계 유형을 나타낸 것으로, 다른 계층에서는 기능을 호출하는 Call 관계를 형성하며, 동일 계층에서는 기능을 이루는 Set 연관관계

표 13 은행 도메인의 시스템 오퍼레이션 클러스터링 결과 표

| 시스템 오퍼레이션 번호 | 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | | | 5 | 5 | 5 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | |
| 3 | | 5 | 5 | | | | 2.1 | 2.1 | | | | 2 | 5 | 4 | 4 | 4 | 2.1 | | | |
| 4 | | 5 | 5 | | | | 0.5 | 0.3 | | | | 2 | 3.5 | 3.5 | 3.5 | 3.5 | 0.3 | | | |
| 5 | 4.5 | | | 5 | 4.3 | 3.3 | 2 | 2 | 2 | 2 | 2 | | 0.5 | 0.5 | 0.5 | 0.5 | | | | |
| 6 | 4.5 | | | 4.3 | 5 | 3.3 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | |
| 7 | 5 | | | 3.3 | 3.3 | 5 | 2 | 2 | 2 | 2 | 2 | | | | | | | | | |
| 8 | 2 | 2.1 | 0.5 | 2 | 2 | 2 | 5 | 3.3 | 2 | 2 | 2 | | 0.5 | | | | 0.3 | | | |
| 9 | 2 | 2.1 | | 2 | 2 | 2 | 3.3 | 5 | 2 | 2 | 2 | | | | | | 0.3 | | | |
| 10 | 2 | | | 2 | 2 | 2 | 2 | 2 | 5 | 3 | 3 | | | | | | | | | |
| 11 | 2 | | | 2 | 2 | 2 | 2 | 2 | 3 | 5 | 3 | | | | | | | | | |
| 12 | 2 | | | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 5 | | | | | | | | | |
| 14 | | 2 | 2 | | | | | | | | | 5 | 2 | 2 | 2 | 2 | | | | |
| 15 | | 5 | 3.5 | | | | 0.5 | | | | | 2 | 5 | 3.5 | 3.5 | 3.5 | | | | |
| 16 | | 4 | 3.5 | | | | 0.5 | | | | | 2 | 3.5 | 5 | 4 | 4 | | | | |
| 17 | | 4 | 3.5 | | | | 0.5 | | | | | 2 | 3.5 | 5 | 4 | 4 | | | | |
| 18 | | 4 | 3.5 | | | | 0.5 | | | | | 2 | 3.5 | 3.5 | 3.5 | 5 | | | | |
| 19 | | 2.1 | 0.3 | | | | 0.3 | 0.3 | | | | | | | | | 5 | 3 | 3 | 2 |
| 20 | | | | | | | | | | | | | | | | | 3 | 5 | 4 | 2 |
| 21 | | | | | | | | | | | | | | | | | 3 | 4 | 5 | 2 |
| 22 | | | | | | | | | | | | | | | | | 2 | 2 | 2 | 5 |

표 14 은행 도메인의 시스템 인터페이스 목록 표

| 시스템 인터페이스 | 시스템 오퍼레이션 | | 기능성 |
|---|---|---|---|
| | 번호 | 이름 | |
| *IS*DAccount() | 01<br>05<br>06<br>07 | searchAccount()<br>searchPWD()<br>withdraw()<br>deposit() | 계좌 입출금에 관련된 기능 제공 |
| *IS*CDAccount() | 08<br>09 | closeAccount()<br>registerAccount() | 계좌 생성 및 비활성화를 위한 기능 제공 |
| … | … | … | … |
| *IS*SMCommodity() | 19<br>20<br>21 | searchCommodity()<br>adjustCommodityID()<br>modifySaleDays() | 은행 상품에 관련된 기능 제공 |
| *IS*RegisterCommodity() | 22 | registerCommodity() | 은행 상품 생성을 위한 기능 제공 |

형성함을 나타낸다. 표 11에서 $IR$<Interface name$_{IS}$>는 시스템 계층의 $IS$<Interface name$_{I}$>가 비즈니스 계층의 $IB$<Interface name$_{I}$>를 요구하며, 이는 다른 계층 의존도 맺으므로 Call 유형의 연관관계를 형성한다.

### 8.2 Activity 4B: 컴포넌트 인터페이스 명세서 작성

인터페이스 명세서는 컴포넌트가 수행할 기능과 인터페이스 사용자가 기대하는 기능을 정확하게 정의하기 위해 필요한 모든 명세물을 한 곳에 모아놓은 산출물이다[16]. Activity 4B는 지금까지 설계된 인터페이스의 명세서를 작성한다.

인터페이스 명세서에서 인터페이스 구성요소는 구문론(Syntactic)과 의미론(Semantic) 부분을 나타낸다[7]. 즉, 인터페이스 버전과 이름, 오퍼레이션 시그네쳐, 사전/사후 및 제약 조건, 인터페이스 기능 설명 등의 요소를 기술하게 된다. 본 논문에서는 기존에 제시된 요소 외에 인터페이스 설계 시에 기준이 되었던 소프트웨어 아키텍쳐 계층과 인터페이스 유형 등을 추가 명세하여 인터페이스 활동 영역과 역할을 명확히 전달한다. 또한 *Required* 인터페이스 경우 다른 인터페이스와의 연관관계를 명세하며, *Customize* 인터페이스 경우 가변성 유형 및 설정 매개체 등을 명세하여 인터페이스마다의 고유한 특징을 기술한다.

## 9. 사례 연구

본 장에서는 은행 시스템을 바탕으로 시스템 및 비즈니스 *Provide* 인터페이스 설계시 기준 및 가중치에 따른 오퍼레이션 클러스터링의 적절성 검증을 위한 사례 연구를 진행한다. 은행 시스템 범주는 회계, 상품 및 고객 관리의 세부 시스템으로 나누어진다. 세부 시스템은 각각의 기능을 위하여 그림 19와 같이 유즈케이스 리스트를 갖는다.

표 12는 시스템 인터페이스 설계를 위해 그림 19의 각 유즈케이스마다 해당하는 시스템 순차도에 참여하는

표 15 비즈니스 오퍼레이션 식별 표

| 순차도 | | 클래스 | |
|---|---|---|---|
| 시스템 오퍼레이션 | 비즈니스 오퍼레이션 | 번호 | 이름 |
| searchAccount() | searchAccount() | 01 | Account |
| searchPWD() | getPWD() | 01 | Account |
| withdraw() | getContractMaxAmount() | 02 | Commodity |
| | getContractMinAmount() | 02 | Commodity |
| | withdraw() | 01 | Account |
| deposit() | getContractMinAmount() | 02 | Commodity |
| | deposit() | 01 | Account |
| closeBusinessUnit() | checkAllTellerSationClosed() | 05 | TellerStation |
| | searchDailyAccountByBusinessUnit() | 01 | Account |
| | getTotalDepositAmount() | 01 | Account |
| | getTotalWithdrawAmount() | 01 | Account |
| | newBusinessUnitAccount() | 01 | Account |
| | closeBusinessUnit() | 06 | BusinessUnit |
| closeTellerStation() | searchDailyAccountByEmployee() | 01 | Account |
| | searchTellerStation() | 05 | TellerStation |
| | checkTellerStationAmount() | 05 | TellerStation |
| | closeTellerStation() | 05 | TellerStation |
| ... | ... | ... | ... |
| searchCommodity() | searchCommodity() | 02 | Commodity |
| adjustCommodityID() | searchCommodity() | 02 | Commodity |

표 16 은행 도메인의 비즈니스 클래스 클러스터링 결과 표

| 비즈니스 오퍼레이션 번호 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 5 | 3.4 | 3.7 | | 3 | 3 | 3 |
| 2 | 3.4 | 4.7 | | | | | |
| 3 | 3.7 | | 5 | 5 | | | |
| 4 | | | 5 | 5 | | | |
| 5 | 3 | | | | 5 | 4.7 | |
| 6 | 3 | | | | 4.7 | 5 | 2.7 |
| 7 | 2.3 | | | | | 2.7 | 4.3 |

시스템 오퍼레이션을 식별하여 정리한 표이다.

먼저, 시스템 인터페이스 설계를 위해 Activity 1B에서 제시된 기준을 적용하여 인터페이스에 할당하기 위한 시스템 오퍼레이션 클러스터링을 수행하며, 결과는 표 13과 같다. 이때, DM9와 CM8은 시스템 내부에서 실행되는 유즈케이스로 클러스터링에서 탈락된다. 또한 본 사례연구에서는 프로젝트 가중치를 1로 하여 가중치 적용에 의한 합계의 변동을 최소화하여, 논문에서 제시한 방법에 따라 획득한 기준 점수가 그대로 클러스터링에 반영되게 하였다. 단, 기준에 부여되는 가중치는 다양한 도메인의 특성에 따라 다르게 적용되어 도메인에서 원하는 인터페이스의 성질을 결정할 수 있다.

각 오퍼레이션의 기준 점수에 가중치를 적용한 합계는 표 13과 같다. 이때, 합계가 0.0을 제외한 최하 0.3점과 최상 5.0점의 중간점을 반올림한 적정 점수 3.0 이상이 되면 클러스터링하여 고응집력의 인터페이스를 추출하게 된다. 표 14는 식별된 은행의 시스템 인터페이스 목록이다.

다음으로, 비즈니스 인터페이스 설계를 위해 표 15와 같이 시스템 순차도를 확장한 순차도에 참여하는 비즈니스 오퍼레이션과 해당하는 클래스를 식별하여 정리하였다.

식별된 클래스는 Activity 2B에서 제시된 기준을 적용하여 클래스 클러스터링을 수행하며, 클러스터링 결과는 표 16과 같다. 이때, 기준에 대한 시스템 인터페이스와 동일하게 프로젝트의 가중치를 1로 적용하였다.

표 16에서 클러스터링을 위한 적정 점수는 합계의 최하 2.3점과 최상 5.0점의 중간점을 반올림한 4.0이 되며, 합계가 적정 점수 이상이 되는 경우만 클러스터링하게 된다. 표 16에서 클러스터링 된 클래스의 오퍼레이션은 클래스군의 기능을 대표하는 비즈니스 인터페이스의 오퍼레이션이 된다. 만일, 시스템 내부에서 수행하는 오퍼레이션일 경우 비즈니스 인터페이스의 오퍼레이션에서 탈락된다. 표 17은 클러스터링을 통해 식별된 비즈니스 인터페이스이다.

표 17 은행 도메인의 비즈니스 인터페이스 목록 표

| 비즈니스 인터페이스 | 클래스 및 오퍼레이션 | | | | 기능성 |
|---|---|---|---|---|---|
| | 핵심클래스 | 관리클래스 | 관리클래스 오퍼레이션 | 탈락된 오퍼레이션 | |
| *IB*Account() | AccountMgt | Account | searchAccount()<br>getPWD()<br>withdraw()<br>deposit()<br>setCloseFlag()<br>newAccount()<br>searchDailyAccountByBusinessUnit()<br>getTotalDepositAmount()<br>getTotalWithdrawAmount()<br>newBusinessUnitAccount()<br>searchDailyAccountByEmployee()<br>newDailyAccount() | getPWD()<br>setCloseFlag()<br>getTotalDepositAmount()<br>getTotalWithdrawAmount() | 계좌 입출금 관련 |
| *IB*Commodity() | CommodityMgt | Commodity | getContractMaxAmount()<br>getContractMinAmount()<br>searchCommodity()<br>modifySaleDays()<br>newCommodity() | getContractMaxAmount()<br>getContractMinAmount() | 은행 상품 관련 |
| ... | ... | ... | ... | ... | ... |

표 18 컴포넌트 인터페이스 설계 기법 비교

S: Support, P: Partial Support, N: Not Supprot

| 평가요소 \ 연구 | | Cheesman 기법 | Catalysis 방법론 | Katharine 기법 | 본 논문의 기법 |
|---|---|---|---|---|---|
| 정의된 인터페이스 유형 | 기능제공 | S | S | S | S |
| | 기능사용 | N | N | S | S |
| | 기능특화 | N | S | S | S |
| 아키텍처계층 | | S | S | S | S |
| 기능 고유성 | | S | S | P | S |
| 기능고응집성 | | S | S | P | S |
| 기능저결합성 | | S | S | P | S |
| 기능 특화성 | | N | S | P | S |
| 기능 명세성 | | P | P | P | S |
| 상세설계지침 | | P | P | N | S |
| 습득용의성 | | P | P | N | P |

## 10. 평 가

본 논문에서 제시한 컴포넌트 인터페이스 기법을 기존 Cheesman 기법, Catalysis 방법론과 Katharine 기법과 비교 평가하면 표 18과 같다.

기존 연구방법은 본 논문에서 정의한 컴포넌트 기능제공 인터페이스를 모두 지원한다. 기능사용 유형의 인터페이스는 Katharine 기법에서 제시하며, Cheesman과 Catalysis는 컴포넌트간 기능 의존도에 대한 표현만 하였다. 기능특화를 위해 Catalysis는 인터페이스 적용 기법을 제시하였으며, Katharine는 인터페이스에 대한 정의는 제공하나 구체적인 설계 기법이 미흡하였다.

다음으로 본 논문의 '3.3. 설계 기준'에서 제시한 기능 고유성, 고응집성, 저결합성, 특화성 및 명세성에 따라 인터페이스 설계 기법을 비교하였다. Cheesman은 다양한 어플리케이션이 동일 기능을 요구할 때 기존 인터페이스를 재사용하고, 연관된 기능은 같은 인터페이스에서 제공하며 인터페이스간 양방향 참조를 피한다. Catalysis 역시 제공하는 역할에 따라 인터페이스를 구분하고, 연관된 기능을 같은 인터페이스에서 제공하며, 기능을 정제하여 인터페이스간 충돌을 피한다. 따라서 Cheesman과 Catalysis는 기능 고유성, 고응집성, 저결합성 모두를 지원한다. 그러나 Katharine은 컴포넌트 재사용 원리를 근거로 하여 기능 고유성, 고응집성, 저결합성에 대한 특성을 부분적으로 설명하였다.

기능 특화성을 위해 Catalysis는 상위 및 하위 인터페이스를 제시하였고, Katharine는 인터페이스를 통한 기능 특화의 이론적인 방법을 부분적으로 제시하였으나, Cheesman에서는 이를 지원하지 않았다. 기능 명세성을

표 19 컴포넌트 인터페이스 설계기준 준수 평가

S: Support, P: Partial Support, N: Not Supprot

| 설계기준 | 준 수 사 항 | | 평 가 |
|---|---|---|---|
| 기 능 고유성 | 시스템 *Provided* | 시스템 오퍼레이션 식별 Step 3<br>시스템 오퍼레이션 클러스터링 (기준 1), (기준 2) | S |
| | 비즈니스 *Provided* | 비즈니스 오퍼레이션 식별 Step 3<br>비즈니스 오퍼레이션 클러스터링 (기준 1), (기준 2) | |
| 기능고응집성 | 시스템 *Provided* | 시스템 오퍼레이션 클러스터링 (기준 3), (기준 4) | S |
| | 비즈니스 *Provided* | 비즈니스 오퍼레이션 클러스터링 (기준 4) | |
| 기능저결합성 | 시스템 *Provided* | 시스템 오퍼레이션 클러스터링 (기준 3), (기준 4) | S |
| | 비즈니스 *Provided* | 비즈니스 오퍼레이션 클러스터링 (기준 4) | |
| 기 능 특화성 | *Customize* | 가변성 유형별 정의 Step 1 | S |
| 기 능 명세성 | 시스템 *Provided* | 컴포넌트 인터페이스 명세서 작성 | S |
| | 비즈니스 *Provided* | 컴포넌트 인터페이스 명세서 작성 | |
| | *Customize* | 컴포넌트 인터페이스 명세서 작성 | |
| | *Required* | *Required* 인터페이스 식별의 인터페이스 연관관계<br>컴포넌트 인터페이스 명세서 작성 | |

위해 Cheesman, Catalysis과 Katharine 모두 인터페이스 기능 명세를 지원한다. 그러나 본 논문에서 제시한 아키텍쳐 계층이나 특화 요소 등의 명세가 미흡하다.

Chessman과 Catalysis 모두 제시된 인터페이스에 대한 상세 설계 지침을 제시하였으나, *Provided*, *Required* 및 *Customize*의 3가지 인터페이스 유형에 대한 부분적인 지침만을 제공하며, Katharine는 상세 지침을 지원하지 않는다. 단, 기존 연구뿐만 아니라 본 논문의 기법에서도 인터페이스 설계시 UML 등의 사전 지식이 필요할 수 있다. 그러나 본 연구에서는 기존 연구보다 인터페이스를 위한 상세한 설계 지침을 제공하여 컴포넌트 인터페이스 설계 기법 도입이 용의할 것이다.

다음으로, '3.3. 설계 기준'에서 제시한 설계기준이 본 논문의 어느 부분에서 준수 되었는가를 평가하면 표 19와 같다.

본 논문에서 정의된 설계기준은 프로세스의 여러 부분에서 준수되었으므로 본 논문에서 제시한 컴포넌트 인터페이스의 설계 기법은 컴포넌트의 높은 재사용성과 확장성 및 유지보수성을 위해 실용적으로 사용할 수 있다.

## 11. 결 론

본 논문에서는 컴포넌트 *Provided*, *Required*와 *Customize* 인터페이스를 위한 체계적인 설계 프로세스를 제시하였다. 제시된 기법은 소프트웨어 아키텍쳐에 따라 시스템 및 비즈니스 계층의 인터페이스로 구분하여 컴포넌트 기능을 제공하는 *Provided*와 제공받는 *Required* 인터페이스를 제시하였으며, 또한 컴포넌트 자체를 수정하지 않고 컴포넌트 배치시 사용자 요구에 맞게 컴포넌트를 특화하는 *Customize* 인터페이스 설계 기법을 제안하였다.

컴포넌트 인트페이스의 3가지 유형은 유즈케이스, 순차도와 클래스 모델 등 UML 기반 산출물을 근거로 인터페이스 설계시 모든 산출물의 추적성이 보장된다. 또한 BNF 형식의 정규언어로 인터페이스를 정의하여 설계자의 이해를 도왔으며, 가변성 및 공통성 오퍼레이션으로 구성된 인터페이스 설계 및 명세를 위한 세부적인 지침을 제공하였다. 컴포넌트 *Provided*, *Required* 및 *Customize* 인터페이스 설계 및 명세를 행함에 따라 향후 재사용 및 확장성 등을 요구하는 어플리케이션을 위한 컴포넌트 인터페이스 설계에 활용될 것으로 기대된다.

## 참 고 문 헌

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

[2] Short, K., Component Based Development and Object Modeling, Sterling Software, Technical Handbool Version 1.0, Februaray 1997.

[3] Chessman, J., and Daniels, J., *UML Component*, Addison-Wesley, 2001.

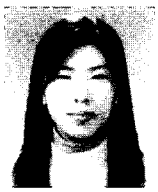
[4] D'Souza, D. S., and Wills, A. C., Objects, Components, and Frameworks with UML: The Catalysis Approach, Addison Wesley, 1999.

[5] Whitehead, K., Component-based Development: Principles and Planning for Business Systems, Addison Wesley, 2002.

[6] FowLer, M., Patterns Of Enterprise Application Architecture, Addison Wesley, 2003.

[7] Szyperski, C., Component Software: Beyound Object-Oriented Programming Second Edition, Addison Wesley, 2002.

[8] Booch, C. K., Jacobson, I., and Rumbaugh, J., *UML Distilied*, Addison Wesley, 2004.
[9] Choi, S. W., Chang, S. H., and Kim, S. D., "A Systematic Methodology for Developing Component Frameworks," Lecture Notes in Computer Science 2984, Proceedings of 7th Fundamental Approaches to Software Engineering (FASE'04) Conference, pp.359-373, 2004.
[10] Michasel, W. B., and Richard, C. C., "A Technique to Identify Component Interfaces," Workshop on Software Architecture Reconstruction at WCRE, November, 2002.
[11] Sterling Software, Identifying and Scoping CBD96 Components, November 1996.
[12] Lee, S. D., Yang, Y. J., Cho, E. S., and Kim, S., D., *COMO: A UML-Based Component Development Methodology*, Asia-Pacific Software Engineering Conference, pp.54-61, December, 1999.
[13] Geyer, L., and Becker, M., "On the Influence of Variabilities on the Application-Engineering Process of a Product Family," SPLC2 2002, LNCS 2379, pp.1-14, 2002.
[14] Kim, C. J., and Kim, S. D., "The Static and Dynamic Customization Technique of Component," Journal of KISS: Software and Applications, Vol.29, No.09, pp.605-618, October 2002.
[15] So, D. S., Shin, S. K., and Kim, S. D., "A Formal Model of Component Variability Types and Scope," Journal of KISS: Software and Applications, Vol. 30, No.05, pp. 414-429, June 2003.
[16] Clements, P., Bachmann, F., and Bass, L., *Documenting Software Architectures*, Addison Wesley, 2003.



박 지 영
1998년 3월~2002년 2월 한세대학교 컴퓨터정보통신공학부(공학사). 2002년 9월~2004년 8월 숭실대학교 컴퓨터학과(공학석사). 관심분야는 컴포넌트 개발 방법론, 소프트웨어 아키텍처, 웹서비스

김 수 동
정보과학회논문지 : 소프트웨어 및 응용
제 31 권 제 1 호 참조