컴포넌트 아키텍쳐 기반의 동적 컴포넌트 조합을 위한 가변성 설계 기법

A Variability Design Technique based on Component Architecture for Dynamic Component Integration

김 철 진* 조 은 숙**
Kim Chul Jin Cho Eun Sook

요 약

컴포넌트 조합을 통한 소프트웨어 개발은 Time-To-Market을 위한 시대적인 흐름이며 소프트웨어의 짧은 생명주기(Lifecycle)를 극복할 수 있는 방안이다. 그러므로 효과적으로 컴포넌트를 통합하기 위한 기법들이 연구되어 왔다. 그러나 체계적이고 실용적인 기법들의 연구가 미흡하였다. 컴포넌트 통합을 위한 핵심 중에 하나는 통합을 위한 스펙을 어떻게 정의하느냐와 그 통합 스펙을 운영하기 위한 컴포넌트 아키텍쳐를 어떻게 구성하느냐 이다.

본 논문은 특화가 가능한 복합 컴포넌트를 개발하기 위해 컴포넌트 아키텍쳐를 기반으로 컴포넌트 간의 조합을 위한 가변성 설계 기법을 제안하며 가변성 중에 컴포넌트 간의 메시지의 흐름인 워크플로우(Workflow)에 대한 가변성 설계 기법을 제안한다. 본 기법은 컴포넌트 아키텍쳐 기반의 연결 계약(Connection Contract)을 설계하는 것이 핵심이다. 연결 계약은 컴포넌트의 제공 인터페이스(Provided Interface)를 사용하여 설계하며 이 연결 계약에 의해 동적으로 컴포넌트를 조합하고 특화(Customiztion)할 수 있는 기법을 제공한다.

Abstract

Software development by component integration is the mainstream for time-to-market and is the solution for overcoming the short lifecycle of software. Therefore, the effective techniques for component integration have been working. However, the systematic and practical technique has not been proposed. One of main issues for component integration is how to specify integration and the component architecture for operating the specification.

In this paper, we propose a workflow variability design technique for component integration. This technique focuses on designing to a connection contract based on the component architecture. The connection contract is designed to use the provided interface of component and the architecture can assemble and customize components by the connection contract dynamically.

☞ Keyword : Component Integration, Connection Contract, Variability, Workflow, Component Architecture

1. 서 론

현 엔터프라이즈 시스템의 비즈니스는 빠르게 진화하고 있으며, 이에 따라 비즈니스 모델의 빈번한 변경을 요구하고 있다. 이러한 잦은 변경은 소프트웨어의 복잡성과 유지보수 비용의 증가를 발생시키기 때문에 고 품질의 재사용 가능한 비즈니스 컴포넌트에 대한 요구가 증가하고 있다[1,2]. 그러나 다양한 도메인의 요구 사항을 위한 재사용 가능한 비즈니스 컴포넌트는 만족스럽게 제공되지 않는 상황이다. 이러한 이유는 컴포넌트를 개발할 때 비즈니스 로직에만 초점을 맞추어 개발하고 재사용성에 초점을 맞추어 개발되고 있지 않기 때문이다.

따라서 본 논문에서는 다양한 도메인에 사용될 수 있도록 재사용 가능한 복합 컴포넌트 설계 기

* 정 회 원 : 가톨릭대학교 컴퓨터 정보 공학부 교수
cjkim@otlab.ssu.ac.kr(제 1저자)
** 정 회 원 : 서일대학 소프트웨어공학과 전임강사
escho@seoil.ac.kr(공동저자)
[2004년/07/20 투고 - 2004/08/13,14,16 심사 - 2004/09/16 심사완료]

법을 제안한다. 컴포넌트 간의 조합은 메시지 흐름의 조합을 통해 가능하기 때문에 메시지 흐름을 설계할 수 있는 워크플로우 가변성 설계 기법을 제안한다. 컴포넌트 간의 가변적인 메시지 흐름을 위해 커넥터(Connector) 기법을 제안하며, 컴포넌트 아키텍쳐가 커넥터 패턴을 따른다.

논문의 구성은 2장에서 컴포넌트 모델 개념과 기존의 가변성 연구들에 대해 소개하고 컴포넌트 통합 시의 한계점을 제시한다. 또한 본 논문의 기본 개념으로 가변성에 대해 정의한다. 3장은 본 논문의 핵심 기법인 워크플로우 가변성에 대한 설계 기법을 제안한다. 4장은 사례 연구를 통해 본 기법의 재사용을 검증한다. 5장에서는 결론을 맺고 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 컴포넌트 모델

컴포넌트 모델은 컴포넌트의 기본적인 아키텍쳐, 컴포넌트의 인터페이스, 그리고 컴포넌트와 컨테이너 간의 상호작용을 위한 메커니즘을 정의한다. 이와 같이 컴포넌트 모델은 재사용할 수 있는 컴포넌트를 지원하기 위한 환경을 정의한다[3]. 컴포넌트는 다른 컴포넌트나 프레임워크와 상호작용할 수 있도록 하기 위해 미리 정의된 아키텍쳐에 맞게 설계되고 구현되어야만 한다. 컴포넌트 기반 아키텍쳐는 컴포넌트를 첨가하거나 대치하여 기능적인 향상을 이룰 수 있도록 프레임워크 형태로 제공된다[4].

컴포넌트는 블랙박스 형태의 재사용 단위로 개발 시스템에서 컴포넌트를 사용하기 위해 컴포넌트 인터페이스만을 알면 되며 내부 구현 모듈을 인식할 필요가 없다. 인터페이스는 컴포넌트에 의해 제공되는 기능을 정의하며 컴포넌트 내의 하나 이상의 클래스들에 의해 논리적으로 구현된다. 컴포넌트 인터페이스는 외부에 기능을 제공하는 역할뿐만 아니라 컴포넌트를 관리하고 제어하는 역할을 한다. 주로, 컴포넌트 사용자에 의해 컴포넌트를 조합하거나 커스터마이제이션하기 위한 기능을 제공할 수 있다. 제공되는 서비스에 한정하여 인터페이스를 정의하면 다양한 도메인에 적용될 때 상이한 요구사항을 충족시키기 위해 컴포넌트 내부를 변경해야하는 문제가 발생할 수 있다.

2.2 CBD 방법론

Component Based Development(CBD) 방법론들은 특화의 필요성에 대해 언급하고 있으며 개념적인 수준에서 접근 방법을 제시하고 있다. 특화에 대해 언급하고 있는 방법론으로 Catalysis와 Componentware는 컴포넌트의 가변성(Variability)을 정의하여 특화에 대해 언급하고 있다.

Desmond F. D'Souza에 의해서 제안된 Catalysis는 1991년에 OMT의 정형화로서 시작 되었으며 객체나 프레임워크와 함께 컴포넌트 기반 개발을 위한 차세대 방법론으로 출현하게 되었다. Catalysis는 모델 간에 일관성 규칙을 잘 정의하고 있으며 복잡한 시스템을 설계하는데 다양한 뷰(View)를 제공하기 위한 기법을 정의하고 있다. 추가적으로 Catalysis는 다른 방법론과는 다르게 프레임워크를 사용하여 컴포넌트를 개발할 수 있는 개발 프로세스를 정의한다[2]. Catalysis의 프로세스는 요구사항 분석, 시스템 스펙, 아키텍쳐 설계, 그리고 컴포넌트 내부 설계 단계로 구성되며 설계부터 소스 코드에 이르는 전과정의 추적성(Traceability)과 정확성(Precision)을 보장하고 재사용 측면에서는 소스 코드뿐만 아니라 설계 산출물까지 재사용할 수 있도록 한다[2]. Catalysis는 프로세스 패턴(Process Pattern)을 정의하여 다양한 프로젝트에 유연하게 프로세스를 구성할 수 있도록 지원한다. Catalysis에서는 컴포넌트를 변경하기 위한 방법으로 "Plug-Point"를 정의하여 여러 도메인에서 공통적인 부분으로 각각 특성화된 로직을 플러그 인할 수 있는 부분을 정의한다. 일반 컴포넌트(Generic Component)는 "Plug-Point"

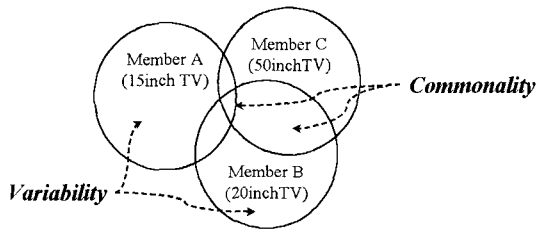
를 정의하고 적용되는 도메인의 요구 사항에 맞는 "Plug-Ins"(모듈이나 컴포넌트)를 플러그 인하여 일반 컴포넌트을 변경할 수 있다. "Plug-Point"는 특정 도메인의 어플리케이션에 사용되기 위해 가변적으로 적용될 수 있도록 하기 때문에 "Variation-Point"라고도 한다. 어플리케이션에 적용되기 전에 일반 컴포넌트는 미리 만들어진 기본 모듈을 적용할 수 있다. Catalysis는 "Required Interface"를 정의하는 절차를 포함하고 있다. 이 인터페이스는 "Plug-Point"를 제공하는 역할을 하며 커스터마이제이션을 위한 가변성 요소가 될 수 있다. 일반 컴포넌트에서 서비스로 제공되는 인터페이스를 "Provided Interface"로 정의한다.

Componentware의 프로세스는 분석, 비즈니스 설계, 기술 설계, 스펙 그리고 구현 단계로 구성되어 있다. Componetware에서는 이러한 단계를 조합하여 다양한 형태의 프로세스를 구성할 수 있도록 프로세스 패턴들을 제공한다[5]. Componentware는 커스터마이제이션을 위해 Catalysis의 "Required Interface"와 유사한 "Import Interface"를 정의한다. 이 인터페이스도 컴포넌트 외부의 다양한 요구사항을 충족시킬 수 있도록 한다. Componentware는 일반적으로 컴포넌트에서 제공되는 인터페이스를 "Export Interface"로 정의하며 외부 컴포넌트에 제공되는 인터페이스를 "Import Interface"로 정의한다[5]. 이와 같이 "Export Interface"와 "Import Interface" 모두는 컴포넌트 사이와 연결을 가능하게 하며 가변성을 제공할 수 있다.

앞에서 언급한 CBD 방법론인 Catalysis나 Componentware는 가변성 설계를 위한 상세한 절차나 기법보다는 개념적인 적용 방법에 대해서 언급하고 있다. 따라서 본 연구에서 컴포넌트 가변성에 대한 구체적인 설계 기법이나 절차에 대해 고려한다.

2.3 가변성

가변성(Variability)은 공통성(Commonality)과 함께 제품군(Product Family)의 요소[6]로서 제품



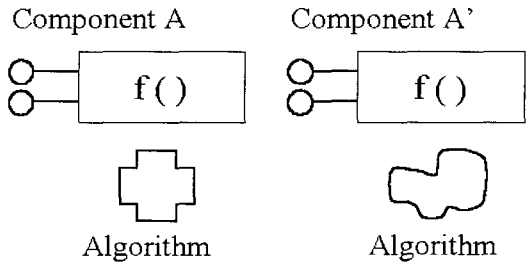
〈그림 1〉 제품군의 가변성과 공통성

군 멤버들 사이의 차이[7]를 말한다.

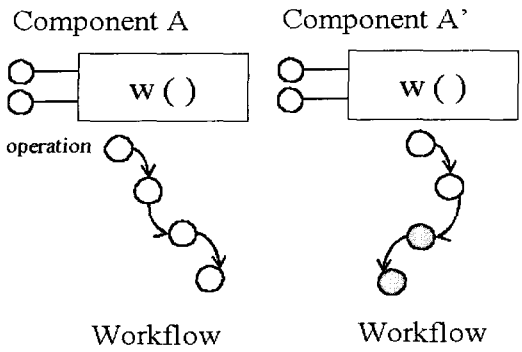
그림 1와 같이 TV 제품군에서 가변성은 화면 사이즈가 되며, 공통성은 화면 분사 방식이나 방송파 수신 방식 등이다. 이와 같이 가변성은 동일 기능을 하는 제품군의 멤버들 간의 차이로서 컴포넌트에서 가변성은 소프트웨어 구성 요소와 일치하는 속성, 오퍼레이션, 그리고 메시지의 흐름이 된다.

행위 가변성은 동일 컴포넌트 멤버들 간에 동일 기능을 하는 오퍼레이션이 서로 다른 알고리즘으로 기능을 제공하는 경우를 행위 가변성이라고 한다. 예를 들면, 은행 도메인의 계좌 컴포넌트에서 이자 계산 기능은 은행 마다 존재하지만 서로 다른 알고리즘으로 기능을 제공하기 때문에 은행마다 이자 계산의 기능은 각각 다르다. 따라서 이자 계산은 행위 가변성에 해당한다. 그림 2와 같이 동일 기능의 컴포넌트 A와 A'가 특정 행위 f()를 위해 서로 다른 알고리즘으로 정의 되는 경우를 의미한다.

워크플로우 가변성은 동일 컴포넌트 멤버들 간에 동일 기능이 서로 다른 메시지 흐름을 제공



〈그림 2〉 행위 가변성



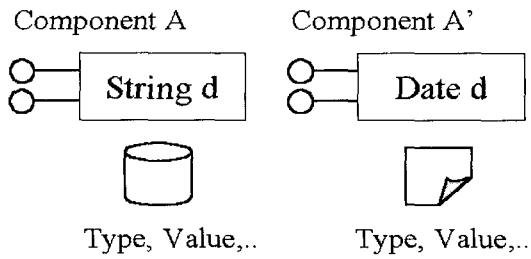
〈그림 3〉 워크플로우 가변성

하는 경우에 워크플로우 가변성이라고 한다. 예를 들면, 회원 등록하는 회원 컴포넌트에서 회원 입력, 회원 인증, 데이터 베이스 저장의 프로세스를 요구할 수 도 있지만 특정 도메인에서는 데이터 베이스 저장이 아니라 레거시 시스템으로 연동되는 프로세스로 요구할 수 있기 때문에 동일한 회원 등록에 대해 다른 워크플로우를 요구하는 경우에 워크플로우 가변성이라고 한다. 그림 3과 같이 동일 기능의 컴포넌트 A와 A'가 특정 행위 w()가 서로 다른 워크플로우의 흐름을 갖는 경우를 의미한다.

속성 가변성은 동일 컴포넌트 멤버들 간에 동일 역할의 속성이 서로 다른 타입이나 값을 가지는 경우 속성 가변성이라고 한다[8]. 예를 들면 동일 컴포넌트에서 날짜를 나타내는 속성이 도메인의 요구 사항에 따라 스트링 타입 이나 데이트 타입으로 요구할 수 있는데 이러한 경우를 속성 가변성이라고 할 수 있다. 그림 4와 같이 동일 기능의 컴포넌트 A와 A'가 특정 데이터 d가 서로 다른 타입으로 정의 되는 경우를 의미한다.

3. 워크플로우 가변성 설계

워크플로우 가변성 설계는 복합 컴포넌트 설계 시 컴포넌트들 간의 메시지 흐름을 특화가 가능하도록 설계하는 것이다. 본 설계 기법은 커넥터(Connector) 구조를 기반으로 컴포넌트들 간의 메

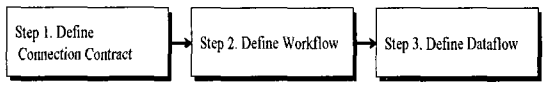


〈그림 4〉 속성 가변성

시지 흐름을 동적으로 특화할 수 있다.

3.1 가변성 설계 프로세스

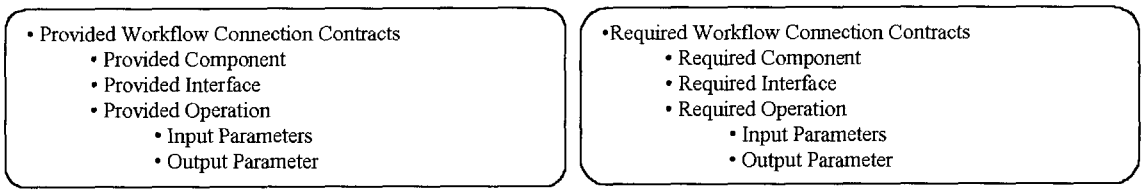
워크플로우 가변성을 위한 복합 컴포넌트 설계 프로세스는 그림 5과 같다. 워크플로우 가변성 설계 프로세스는 연결 계약 정의, 워크플로우 정의, 그리고 데이터플로우 정의 단계로 구성된다. 첫 단계는 상호 작용 표준[6]으로서 연결 계약을 정의한다. 다음 단계는 연결 계약의 맵핑(Mapping)을 통해 워크플로우를 정의한다. 마지막 단계로 연결 계약 시 오퍼레이션의 입력 데이터를 위한 데이터플로우를 정의한다.



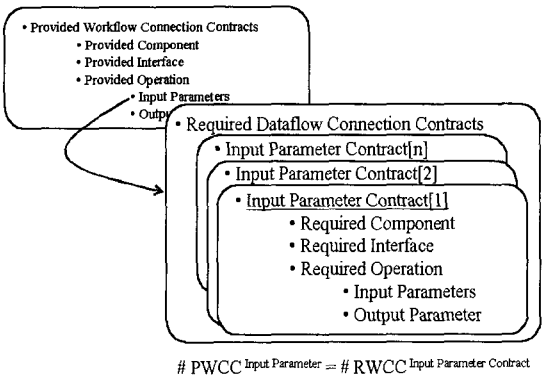
〈그림 5〉 워크플로우 가변성 설계 프로세스

Step 1. Define Connection Contract

본 단계는 컴포넌트들 간에 메시지 흐름이나 데이터 흐름을 연결하기 위한 계약(Contract)을 정의하는 단계이다. 연결 계약의 종류는 3가지로 Provided Workflow Connection Contracts(PWCC), Required Workflow Connection Contracts(RWCC), Required Dataflow Connection Contracts(RDCC)로 구성된다. PWCC는 현 흐름에 있는 오퍼레이션에 대한 정보로서 그림 6과 같이 컴포넌트 명, 인터페이스 명, 오퍼레이션 명과 오퍼레이션 파라메터로 구성된다. RWCC의 정보도 PWCC와 유



〈그림 6〉 PWCC와 RWCC

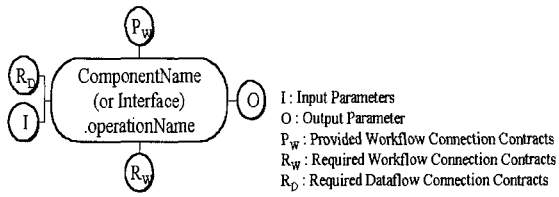


〈그림 7〉 RDCC

사한 정보로 현 오퍼레이션 다음에 호출해야 할 오퍼레이션의 정보를 포함하고 있다.

RDCC는 현 오퍼레이션이 처리하기 위해 필요한 입력 파라메터를 어디로부터 얻어와야 할지에 대한 정보를 포함하고 있는 연결 계약이다. 그림 7와 같이 PWCC에 정의된 입력 파라메터 각각에 대해 어느 오퍼레이션으로부터 얻어와야 할지에 대한 정보를 포함한다.

3개의 연결 계약을 기본으로 워크플로우와 데이터 흐름을 설계할 수 있다. 이러한 흐름 설계는 연결 계약의 맵핑(Mapping)을 통해 이루어 질 수 있다. 본 논문에서는 흐름 설계를 위해 그림 8과



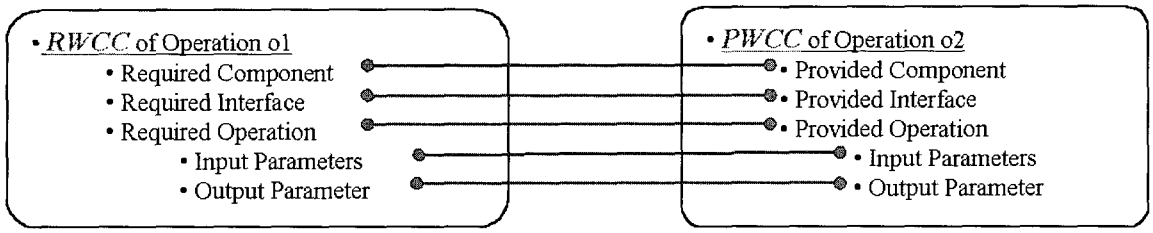
〈그림 8〉 오퍼래이션 표기법

같은 표기를 정의 한다.

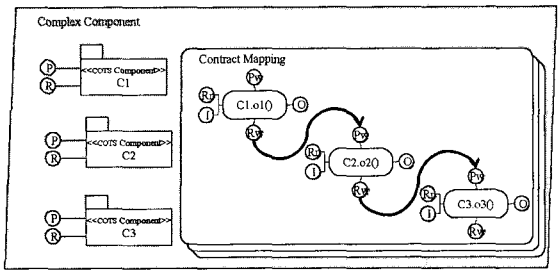
Step 2. Define Workflow

본 단계에서는 정의된 연결 계약을 기본으로 컴포넌트들 간의 흐름을 정의하는 단계이다. 워크플로우는 현재 오퍼레이션의 RWCC와 다른 오퍼레이션의 PWCC의 맵핑을 통해 정의한다. 그림 9와 같이 RWCC의 컴포넌트(Required Component), 인터페이스(Required Interface), 오퍼레이션(Required Operation)의 스펙과 일치하는 다른 컴포넌트의 PWCC와의 맵핑을 통해 워크플로우를 정의한다.

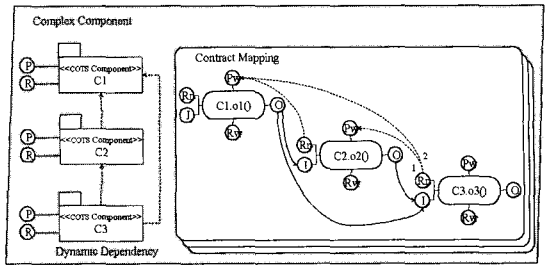
즉, 현재 오퍼레이션에서 요구하는 오퍼레이션 연결 계약(RWCC)을 기본으로 다른 오퍼레이션들의 연결 계약(PWCC)과의 맵핑을 통해 일치하는 경우 연결을 정의한다. 이렇게 정의된 연결 계약 정보는 커넥터에 의해 동적으로 순차적으로 호출



〈그림 9〉 워크플로우를 위한 연결 계약 맵핑



〈그림 10〉 워크플로우 설계



〈그림 12〉 데이터 플로우 설계

될 수 있다.

복합 컴포넌트의 워크플로우는 그림 10과 같이 설계 될 수 있다. 컴포넌트들 간의 메시지의 흐름은 오퍼레이션들 간의 순차적인 호출을 의미한다. 워크플로우 설계에서 오퍼레이션 o1의 호출 후에 호출될 오퍼레이션의 정보는 o1의 RWCC에 정의되어 있으며 o1의 RWCC와 일치하는 다른 오퍼레이션의 PWCC를 연결하여 다음으로 호출될 오퍼레이션을 정의한다. 이런 방법으로 다른 컴포넌트들 간에 워크플로우를 설계될 수 있다.

Step 3. Define Dataflow

본 단계에서는 워크플로우와 관련 없이 오퍼레이션들이 필요로 하는 입력 데이터를 얻기 위한 데이터 간의 맵핑을 정의하는 단계이다. 데이터 플로우는 현재 오퍼레이션의 RDCC와 일치하는 PWCC를 가진 오퍼레이션과의 맵핑을 통해 데이터 연결을 정의한다.

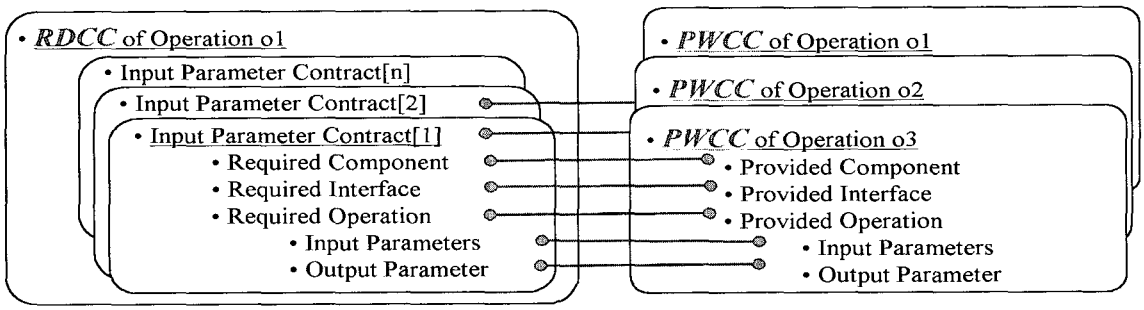
그림 11과 같이 현재 오퍼레이션이 여러 파라메터를 포함할 수 있기 때문에 RDCC는 입력 파라메터 개수의 Input Parameter Contract을 포함한다. 이러한 각각의 연결 계약과 일치하는 PWCC를 가진 다른 오퍼레이션들과의 맵핑을 통해 해당하는 오퍼레이션으로부터 데이터를 얻을 수 있도록 정의한다.

복합 컴포넌트를 위한 데이터 간의 흐름 설계는 그림 12와 같이 설계할 수 있다. 데이터 흐름 설계는 입력 파라메터의 순서를 따라야 하며 이러한 흐름에 의해 컴포넌트들 간의 동적인 의존성(Dynamic Dependency)을 형성한다.

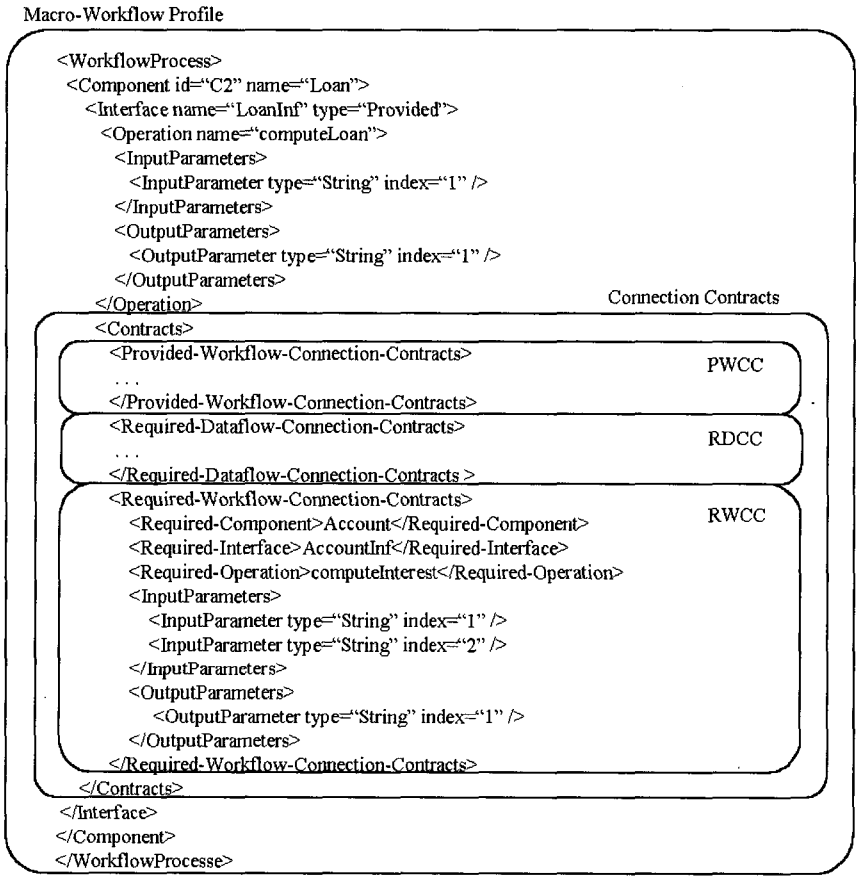
오퍼레이션 o3의 입력 파라메터는 오퍼레이션 o1과 o2의 처리 결과값에서 얻어와야 함을 정의한다. 이때 o3의 RDCC와 일치하는 PWCC를 가진 오퍼레이션을 순차적으로 연결하여 정의한다.

3.2 컴포넌트 조합을 위한 컴포넌트 아키텍쳐

워크플로우와 데이터 플로우의 설계를 통해 생성된 연결 계약은 커넥터에 의해 운영될 수 있도



〈그림 11〉 데이터 플로우를 위한 연결 계약 맵핑



〈그림 13〉 XML 형태의 연결 계약 사례

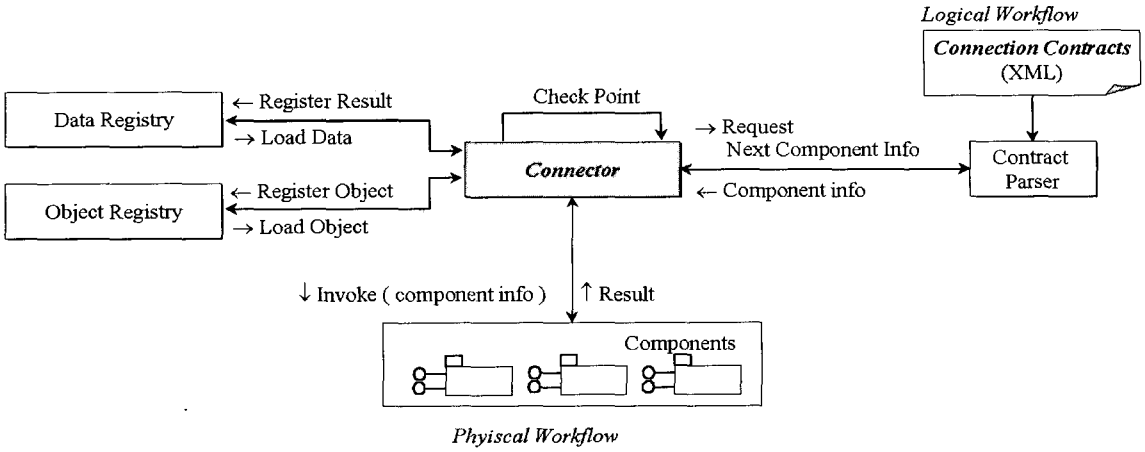
록 구체화된 형태로 정의할 수 있다. 예를 들면 그림 13과 같이 XML 형태로 정의되어 동적으로 관리될 수 있다.

그림 13과 같이 XML 형태의 연결 계약은 실제적으로 커넥터를 통해 복합 컴포넌트를 구성하는 컴포넌트들의 오퍼레이션을 호출하기 위한 정보들을 포함하고 있다. 또한 처리된 결과 데이터를 통해 다른 오퍼레이션의 입력 데이터로 이용될 수 있도록 맵핑 연결 계약도 정의하고 있다. 복합 컴포넌트는 이러한 연결 계약 정보를 통해 운영될 수 있도록 커넥터를 기반으로 한다.

그림 14과 같이 커넥터(Connector)는 동적으로 컴포넌트를 호출하기 위해 각 언어의 메타 호출 기법인 RTTI(Reflection)에 기반한다[9]. 커넥터는 XML 형태의 연결 계약을 입력으로 물리적인 컴포넌트를 호출하며 처리된 결과는 데이터 저장소(Data Registry)에 저장하여 다른 컴포넌트에서 입력으로 요구할 경우 제공한다. 이러한 흐름을 반복으로 처리하여 복합 컴포넌트 내의 컴포넌트들 간의 메시지 흐름을 제어 한다. 워크플로우나 데이터 흐름을 특화하기 위해서는 연결 계약을 변경하여 커넥터에 의해 물리적인 호출을 변경하도록 한다.

4. 사례 연구

본 장에서는 제안된 가변성 설계 기법을 통해 설계된 컴포넌트의 재사용성이 기존 CBD 프로세



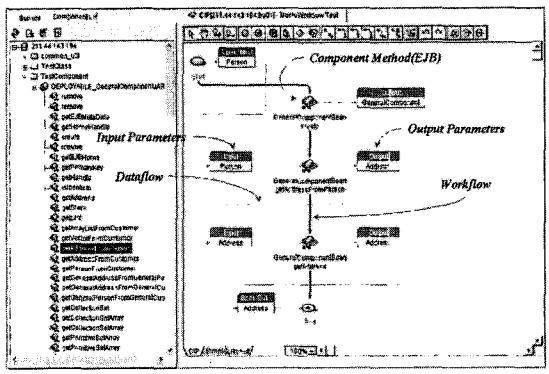
〈그림 14〉 커넥터 아키텍쳐

스를 통해 개발된 컴포넌트 보다 향상 됨을 검증한다. 사례 연구는 특화 사례 조사를 통해 가변성을 추출하였으며, 추출된 가변성에 대해 본 논문에서 제안한 기법을 통해 해당 컴포넌트를 재 설계하였다. 최종적으로, 컴포넌트의 재 개발에 참여한 개발자들을 대상으로 국제 소프트웨어 품질 측정 메트릭인 ISO 9126[11]을 이용해 재사용성의 품질을 측정하였다.

사례 연구의 모집단은 이네트라는 회사의 30명의 사이트 개발자와 일반 개발자를 대상으로 실험하였으며, 본 기법을 이용하여 가변성을 적용한 개발자는 최종적으로 23명이다. 최종 23명은 가변성 추출과 컴포넌트 재 설계 및 재 개발, 그리고 재사용성을 위한 설문 조사에 응한 집단이다.

4.1 가변성 설계 및 적용

워크플로우 가변성은 연결 계약을 통해 워크플로우와 데이터 플로우를 설정하는 단계로 이루어진다. 이와 같은 연결 계약을 설정하기 위해 그림 15과 같은 도구를 이용하여 자동 생성한다. 본 도구는 J2EE 기반의 Java Class나 EJB 컴포넌트를 통합하기 위한 자동화 도구로 개발되었다[10]. 본 사례는 회원 등록을 위한 흐름 중에 주소 등록에 필요한 컴포넌트를 이용하는 흐름을 이용한다.



〈그림 15〉 연결 계약 자동 생성 도구

Step 1. Define Connection Contract

연결 계약은 컴포넌트 내의 오퍼레이션을 정의하는 단계로서 GeneralComponentBean의 create(), getAddressFromPerson(), getAddress() 오퍼레이션에 대한 PWCC를 정의한다.

Step 2. 워크플로우 설계

본 단계에서는 GeneralComponentBean의 create()와 getAddressFromPerson()간의 흐름, getAddressFrom Person()과 getAddress()간의 흐름을 설계하기 위해 create()의 연결 계약인 RWCC에 getAddressFromPerson() 연결 계약의 PWCC와 동일한 정보를 정의한다. 또한 getAddressFrom-

Person()의 연결 계약인 RWCC에 getAddress() 연결 계약의 PWCC와 동일한 정보를 정의한다.

Step 3. 데이터 플로우 설계

본 단계에서는 GeneralComponentBean의 create(), getAddressFromPerson(), getAddress() 오퍼레이션을 실행하기 위해 필요한 입력 매개변수에 대한 정보를 설정하는 단계로서 getAddress()의 입력 매개변수인 Address 객체는 getAddressFromPerson()의 출력 매개변수로부터 얻는다. 이와 같은 정보는 getAddress()의 RDCC와 일치하는 getAddressFromPerson()의 PWCC 정보를 통해 정의된다.

4.2 적용된 가변성 조사

본 단계에서는 컴포넌트의 재사용성을 검증하기 위해 본 논문에서 제안한 기법으로 개발된 컴포넌트와 기존의 CBD 프로세스를 통해 개발된 컴포넌트를 대상으로 조사한다. 조사 대상자는 두 경우 모두 개발에 참여한 개발자를 대상으로 조사되었다. 조사 리스트는 국제 소프트웨어 품질 측정 메트릭인 ISO 9126[11]을 이용하였다.

표 1에서 보는 것과 같이 소프트웨어 품질 측정 메트릭[11-13] 중에 재사용성과 관련된 메트릭으로서 이해성(Understandability), 변경성(Changeability), 교체성(Replaceability), 확장성(Extensibility), 일반성(Generality)를 기본으로 하여 다음과 같은 리스트를 통해 재사용성을 조사하였다.

<Understandability>

U1.1 본 기법을 적용하여 컴포넌트 가변성을 설계 할 때, 이해가 쉽습니까 ?
예 ( ) 아니오 ( )
U1.2 기존의 방법으로 컴포넌트 가변성을 설계 할 때, 이해가 쉽습니까 ?
(설계 기법이 존재하지 않았으면 '아니오'로 응답하시면 됩니다.)
예 ( ) 아니오 ( )
U2.1 본 기법을 적용하여 커스터마이제이션 할 때, 가변성 스팩을 통해 이해가 가능합니까 ?
예 ( ) 아니오 ( )
U2.2 기존의 방법으로 커스터마이제이션 할 때, 가변성 스팩을 통해 이해가 가능합니까 ?
(가변성 스팩이 제공되지 않으면 '아니오'로 응답하시면 됩니다.)
예 ( ) 아니오 ( )
U3.1 본 기법을 적용하여 커스터마이제이션 할 때, 가변성 스팩에 해당 도메인에서 요구하는 데이터가 존재합니까 ?
예 ( ) 아니오 ( )
U3.2 기존의 방법으로 커스터마이제이션 할 때, 가변성 스팩에 해당 도메인에서 요구하는 데이터가 존재합니까 ?
(가변성 스팩이 제공되지 않으면 '아니오'로 응답하시면 됩니다.
요구하는 데이터를 생성할 수 있는 스팩이 제공된다면 '예'로 응답하시면 됩니다.)
예 ( ) 아니오 ( )
U4.1 본 기법을 적용하여 커스터마이제이션 할 때, 입력과 출력에 대한 이해가 가능합니까 ?
예 ( ) 아니오 ( )
U4.2 기존의 방법으로 커스터마이제이션 할 때, 입력과 출력에 대한 이해가 가능합니까 ?
(가변성 스팩이 제공되지 않으면 '아니오'로 응답하시면 됩니다.)
예 ( ) 아니오 ( )

〈그림 16〉 이해성 조사 리스트

그림 16의 이해성 조사 리스트처럼 변경성, 교체성, 확장성, 그리고 일반성 조사 리스트도 작성되었다. 위의 조사 리스트는 ISO 9126의 메트릭에 요소들을 기반으로 작성된 것이며, 조사 리스트 번호 앞에 E가 붙은 것은 추가된 질문 리스트이다.

각각의 측정 메트릭에 의해 조사된 결과는 다음과 같다.

표 2는 그림 16의 조사 리스트를 기반으로 조사된 결과이며, 조사 리스트에 대해 긍정적인 대답과 부정적인 대답에 대한 통계적인 수치를 나타낸다. 변경성, 교체성, 확장성, 일반성도 위와 같은 방식으로 조사되었다.

〈표 1〉 소프트웨어 품질 측정 메트릭(ISO9126)

Quality attribute	Sub-Characteristic			
Functionality	Suitability	Accuracy	Inter-operability	Security
Reliability	Maturity	Fault tolerance	Recoverability	
Usability	*Understandability*	Learnability	Operability	Attractiveness
Efficiency	Time behavior	Resource utilization		
Maintainability	Analyzability	*Changeability*	Stability	Testability
Portability	Adaptability	Installability	Co-existence	*Replaceability*

〈표 2〉 메트릭 조사 결과

Understandability(Workflow)				
Survey	U1	U2	U3	U4
NM	23	23	23	23
NP	15	14	12	14
NPE	3	2	0	2
NPR	0.6521739	0.6086957	0.5217391	0.6086957
NPER	0.1304348	0.0869565	0	0.0869565

Changeability(Workflow)			
Survey	C1	C2	EC3
NM	23	23	23
NP	11	7	9
NPE	0	0	0
NPR	0.4782609	0.3043478	0.3913043
NPER	0	0	0

Replaceability(Workflow)				
	R1	R2	ER3	ER4
NM	23	23	23	23
NP	12	13	9	11
NPE	1	0	0	0
NPR	0.5217391	0.5652174	0.3913043	0.4782609
NPER	0.0434783	0	0	0

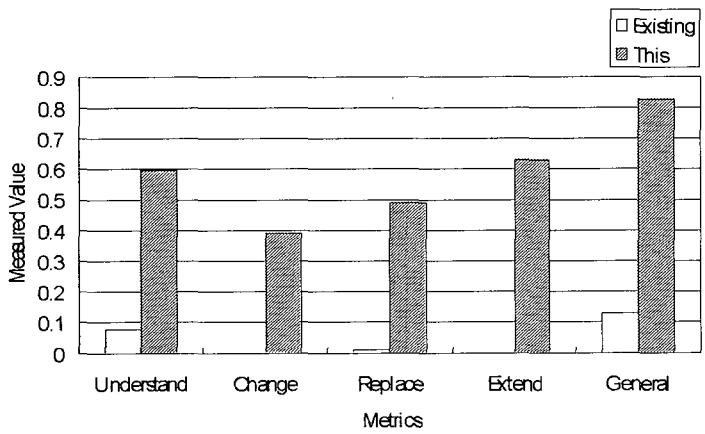
Extensibility(Workflow)		
Survey	EE1	EE2
NM	23	23
NP	15	14
NPE	0	0
NPR	0.6521739	0.6086957
NPER	0	0

Generality(Workflow)	
Survey	EG1
NM	23
NP	19
NPE	3
NPR	0.826087
NPER	0.1304348

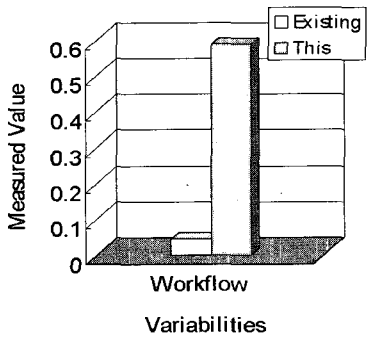
NM : Number of Measurements
NP : Number of Positive Responses of This Method
NPE : Number of Positive Responses of Existing Method
NPR : Positive Responses Ratio of This Method
NPER : Positive Responses Ration of Existing Method

5가지 메트릭에 대한 측정 결과는 그림 17 (1)과 같이 그래프로 나타낼 수 있다. 이해성 메트릭의 측정 결과는 기존 기법에 비해 본 기법의 이해성이 워크플로우 설계 문서를 통해 쉽게 이해될 수 있음을 나타낸다. 기존의 CBD 방법론에서의 순차도는 컴포넌트들 간의 데이터간의 흐름까지 명시하지 못하므로 정확한 관계성을 이해하는데 어려움이 있다. 변경성의 측정 결과는 기존의 기법이 거의 변경하기 위한 가변성을 제공하고 있지 않기 때문에 현격하게 향상됨을 알 수 있다.

교체성은 컴포넌트가 경량으로 다양한 도메인에 적용되기 위해서 필요한 요소로서 교체 시 컴포넌트 내의 다른 부분들과의 상호 연동성도 제공할 수 있어야 한다. 본 측정 결과를 통해 기존 CBD



(1) Measurement Graphs of Metrics



(2) Average Graphs of Metrics

〈그림 17〉 재사용성 하위 메트릭들의 측정 그래프

방법론에서는 워크플로우를 위한 변경이나 교체, 그리고 확장을 위한 기법 제시 않고 있음을 알 수 있다.

그림 17의 (2)는 5가지 메트릭의 측정 결과에 대한 평균 그래프로서, 본 기법을 통해 재설계된 컴포넌트의 재사용성이 향상됨을 알 수 있다. 유효 수준을 측정 결과값 > 0.5로 가정 했을 때, 귀무가설에 따라 워크플로우 가변성 설계 기법에 의한 컴포넌트의 재사용성이 향상됨을 알 수 있다.

5. 결 론

본 논문은 재사용성을 향상 시켜주기 위한 워크플로우 가변성을 위한 컴포넌트 설계 기법을 제안하였다. 기존의 기법들이 개념적인 측면에서 가변성을 접근하고 있는 것과 대조적으로 본 논문에서는 설계 절차를 제안하여 구체적인 기법을 제안하였다. 또한 제안된 기법을 통해 컴포넌트의 재사용성이 향상됨을 살펴보았다.

소프트웨어 개발 패러다임은 소프트웨어 개발의 생산성을 향상 시켜주기 위해 객체 지향 패러다임에서 CBD 또는 PLE(Product Line Engineering)로 진화했지만, 소프트웨어 개발의 생산성은 혁신적으로 향상되지 못했다. 그 이유는 재사용 가능한 비즈니스 컴포넌트가 만족할 만큼 제공되지 못하기 때문일 것이다. 또 다른 측면에서, 고품질의 재사용 가능한 컴포넌트를 개발하기 위한 기법들의 연구가 미흡하기 때문일 것이다. 따라서 본 논문에서는 컴포넌트의 재사용성을 향상 시켜 궁극적으로는 소프트웨어 개발의 생산성을 향상시키고자 한다.

향후 연구 과제는 내장 소프트웨어(Embedded Software)의 가변성 설계 기법에 대해 연구한다. 내장 소프트웨어는 특정 아키텍쳐에 종속되어 개발되지 않고 다양한 아키텍쳐에서 운영될 수 있도록 개발되어야 하며 분산 환경이나 유비쿼터스(Ubiquitous) 환경에서 운영될 수 있도록 개발되어야 한다. 이와 같이 내장 소프트웨어의 적응성을 제공하기 위해서는 특화될 수 있도록 설계되고 개발되어야 한다. 따라서 내장 소프트웨어 개발 시 변경될 가능성이 있는 부분을 설계하기 위한 가변성(Variability) 설계 기법을 연구한다.

참 고 문 헌

[1] Kang K: Issues in Component-Based Software Engineering, 1999 International Workshop on Component-Based Software Engineering.

[2] D'Souza, D. F., and Wills, A. C., Objects, Components, and Frameworks with UML, Addison Wesley Longman, Inc., 1999.

[3] Mikio A., "New Age of Software Development : How Component-Based Software Engineering Changes the Way of Software Development", International Workshop on Component-Based Software Engineering 1998.

[4] Digre T., "Business Object Component Architecture," *IEEE Software*, pp.60-69, September 1998.

[5] Rausch A. *"Software Evolution in COMPONENTWARE Using Requirements/Assurances Contracts"*, Proceedings of the 22th International Conference on Software Engineering, 06/2000

[6] Heineman, G. T. and Councill, W. T., "Component-Based Software Engineering", Addison-Wesley, 2001.

[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, Pearson Education Ltd, 2002.

[8] Kim C. J. and Kim S. D., "A Component Workflow Customization Technique", Vol.27, No.5, Korea Information Science Society, 2000.

[9] JavaWorld webzine, http://www.javaworld.com.

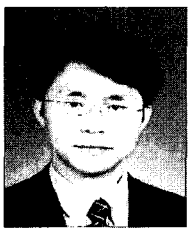
[10] A-WORKS(AutomationWare WORKS™), http://www.enet.co.kr/enet/korean/products/pro_automation.jsp.

[11] ISO/IEC JTC1/SC7 N2419 "DTR 9126-2: Software Engineering - Product Quality Part 2 - External Metrics", 2001.

[12] Kim S. D. and Park J. H., "C-QM: A Practical Quality Model for Evaluating COTS Components", IASTED, SE 2003.

[13] Jeffrey S. P., "Measuring Software Reusability", IEEE Software, 1994.

◐ 저 자 소 개 ◑



김 철 진 (Kim Chul Jin)

1996년 경기대학교 전자계산학과 졸업(학사)
1998년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2004년 숭실대학교 대학원 컴퓨터학과 졸업(박사)
2004년~현재 가톨릭대학교 컴퓨터 정보 공학부 교수
관심분야 : 컴포넌트 기반 소프트웨어 공학, 컴포넌트 커스터마이제이션, 개발 방법론
E-mail : cjkim@otlab.ssu.ac.kr

조 은 숙 (Cho Eun Sook)

1993년 동의대학교 전산통계학과 졸업(학사)
1996년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2000년 숭실대학교 대학원 컴퓨터학과 졸업(박사)
2002년~2003년 한국전자통신연구원 초빙연구원
2000년~현재 서일대학 소프트웨어공학과 전임강사
관심분야 : 개발 방법론, 컴포넌트 기반 소프트웨어 공학, 소프트웨어 아키텍처, 유비쿼터스 컴퓨팅 etc.
E-mail : escho@seoil.ac.kr