소프트웨어 컴포넌트 개발을 위한논리 및 워크플로우 가변성 설계 기법

(Techniques for Designing Logic and Workflow Variability in Software Component Development)

정 광 선 † 김 수 동 ††
(Kwang Sun Jung) (Soo Dong Kim)

요 약 소프트웨어 컴포넌트는 단일 시스템에서 사용하기 보다는 다수의 시스템 혹은 기업, 프로젝트 간에 재사용하기 위한 소프트웨어 모듈이다. 컴포넌트가 다양한 시스템에서 재사용되기 위해서는 여러 시스템에서 요구되는 공통적인 기능을 제공 하는 것뿐만이 아니라 업무상에서 나타나는 다양한 가변적 측면들이 고려되어 고객의 요구에 맞도록 커스터마이즈 될 수 있어야 한다. 사용자가 컴포넌트를 쉽게 사용하기 위해서는 개발 단계에서부터 가변적인 측면이 고려된 컴포넌트가 설계되어 구현되어야 한다. 가변성을 고려하여 쉽게 커스터마이즈 될 수 있는 컴포넌트는 여러 어플리케이션에서 높은 재사용성을 가지게 될 것이다.

기존에 제시된 커스터마이즈 기법들은 소프트웨어가 가지는 가변적인 요소를 어떻게 설계할지에 대해 객체 지향에 기반하여 개념적인 방법만을 제시하고 있으며, 컴포넌트에 적합한 가변성을 고려하고 있는 기법은 드문 실정이다. 따라서 개발이 완료되어 배포된 블랙 박스 형태의 컴포넌트를 커스터마이즈 하기에는 적합하지 않다. 본 논문에서는 컴포넌트가 가지는 기능적인 측면에서의 가변성을 논리와 워크플로우 두 가지로 분류하여 각각의 가변성을 설계하기 위한 선택형, 플러그인, 외부화 세 가지 기법들을 제시한다. 또한 실용적으로 기법들을 적용하기 위해 상세한 설계 지침과 적용 지침들도 제시한다.

키워드 : 컴포넌트, 재사용, 가변성 설계 기법, 커스터마이즈, Required 인터페이스

Abstract A Software Component is a module that is reused among a lot of projects, systems, and companies rather than a single application. Components can be reused in various systems if they provide not only the common functionalities required in many applications but also the diverse aspects to be customized for being suitable for customers' demands. From the development phase, components should be designed and developed considering the variable aspects they have for convenient customization. Easily customized components can be frequently reused in lots of applications.

In the literature, there are some modeling and customizing techniques. But they suggested only conceptual or basic methods based on Object-Oriented. And the practical instructions for reusing component were not provided sufficiently. Moreover, there are few techniques that consider the proper variability types components have. Thus, those techniques are not appropriate for applying to black box component completely developed and released. In this paper, we classify variabilities that components have in functional aspect into two categories. The one is logic variability, and the other is workflow variability. For each classified variability, we propose the three kind of modeling techniques, which are selection, plug in and externalization. Also detailed instructions for practical design and application are provided.

Key words : Component, Reuse, Variability Modeling Techniques, Customize, Required Interface

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 비 회 원 : 숭실대학교 컴퓨터학과
ksjung@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2003년 9월 26일
심사완료 : 2004년 6월 16일

1. 서 론

1.1 동기 및 배경

소프트웨어 어플리케이션을 개발하는데 있어서 개발 노력을 줄이고 Time to Market을 달성하기 위해서 CBD가 등장하게 되었다. 이는 소프트웨어 컴포넌트가

Object-Oriented Programming(OOP)에서의 클래스 보다 더 큰 재사용 단위를 제공하고, 인터페이스만을 통해 서비스를 제공함으로써 재사용성에 용이하기 때문에 어플리케이션을 구축하는 단일 블록으로서 사용되기에 적합하기 때문이다. CBSE[1]뿐만 아니라 PLE[2]에서도 소프트웨어 컴포넌트는 어플리케이션을 구축하기 위한 필수적인 모듈로 사용되고 있다. 재사용성이 높은 컴포넌트를 개발하기 위해서는 동일한 도메인에 속하는 여러 멤버들간의 요구 사항들에 대해 공통성과 가변성(Commonality and Variability : C&V) 분석을 하여 여러 어플리케이션에서 공통적으로 요구되는 요구 사항을 식별하여야 하며, 이렇게 식별된 공통적인 요구 사항을 대상으로 컴포넌트를 개발하게 되면 컴포넌트가 매우 높은 재사용성을 가지게 된다.

컴포넌트는 큰 단위의 서비스를 제공하며 이에 따라 알고리즘뿐만 아니라 업무 흐름까지 반영한다. 따라서 특정 컴포넌트 소비자들이 각각 개발하고자 하는 어플리케이션에 컴포넌트를 사용하려고 할 때, 소비자들이 원하는 기능을 제공하고는 있지만 컴포넌트가 가지는 업무 흐름이나 알고리즘 등이 소비자들이 적용할 업무와는 차이를 가질 수 있다. 컴포넌트 소비자들이 동일한 업무 흐름과 알고리즘을 가진다면 문제가 없겠지만, 업무 알고리즘이나 흐름은 각 조직의 고유한 지적 자산이기 때문에, 산업계에 표준으로서 제공되기에는 현실적으로 어려운 부분이 있다. 따라서 컴포넌트는 동일 도메인 상의 여러 멤버들이 요구하는 다양한 업무 알고리즘과 흐름을 지원할 수 있도록 개발되어야 한다. 이를 위해서는 도메인 내에 존재하는 가변적 부분들에 관한 분석을 통해 가변성을 식별하고, 컴포넌트를 개발 할 때 가변성을 고려하여 설계, 구현해야 한다.

현재 가변성에 대한 연구가 활발이 이루어지고 있으며, 식별 및 분석, 그리고 설계에 대해 여러 가지 기법들이 제시되고 있다. 그러나 제시된 기법들이 개념적인 수준의 기법이거나, 컴포넌트가 가지는 가변성을 구체적으로 고려하지 않은 일반적인 기법을 제시하고 있기 때문에, 컴포넌트가 가지는 가변성에 실용적으로 적용하기에는 미흡한 부분이 있다.

본 논문에서는 우선 컴포넌트가 가지는 기능적 측면의 가변성을 두 가지로 분류한다. 분류된 컴포넌트의 가변성은 제시될 기법을 적용할 대상이 된다. 분류된 가변성을 기반으로 각각에 대한 설계 기법들을 제시하며, 실용적으로 적용할 수 있도록 상세한 지침들을 제시한다.

1.2 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 컴포넌트의 가변성에 대한 설계 기법과 관련된 기존 연구들을 살펴본다. 3장에서는 컴포넌트의 가변성과 컴포넌트의 가변성 분류 및 유형, 그리고 컴포넌트의 인터페이스에 관련된 기반 연구를 다룰 것이며, 4장에서는 분류된 가변성에 대한 설계 기법을 제시한다. 5장에서는 관련 연구에서 다룬 기존 논문들과 본 논문에서 제시된 기법들을 비교 평가하고 6장에서 논문의 결론을 제시한다.

2. 관련연구

2.1 Catalysis의 가변성 구현 기법

Catalysis에서는 프레임워크 상에서 가지는 가변적인 기능을 위해 상속과 템플릿 메소드 기법과 다형성과 전달(Forwarding) 기법을 제시하고 있다[3]. 우선 상속과 템플릿 메소드 기법은 그림 1에서 볼 수 있듯이 가변적인 형태가 다중적인 경우에 기본(Base) 클래스에 대해 다양한 하위 클래스들을 두어 구현하는 방법이다. 이 패턴을 통해 만들어진 소프트웨어는 가변성에 대한 요구사항이 변하게 되면 또 다시 상위 클래스를 상속 받아 구현해야 하기 때문에 한번 이상은 적용하기 힘들다.

다형성과 전달형 방법은 그림 2에서 볼 수 있듯이, 상속과 템플릿 방법에서의 단점을 보완하여 전달 되는 객체에 따라서 기능 수행시간에 동적으로 원하는 기능을 수행하도록 한다. 이를 통해 하나의 컴포넌트가 여러 다른 컴포넌트와 연결될 수 있도록 하고 잘 선택된 집합의 컴포넌트들로 시스템을 구축해 줄 수가 있게 된다.

Catalysis에서 제시된 두 가지 방법은 하나의 클래스 내부에서 가변성을 구현하기 위한 원리적인 방법을 제시해주고 있다.

```
class Hotel{
        public void check_in (Guest g)
        {   this.allocateRoom (g);   }
        protected abstract Room allocateRoom (Guest g);
}

class LeastUsedAllocaingHotel extends Hotel(Guest g);
{
        public Room allocateRoom (Guest g) {   .}
}
```

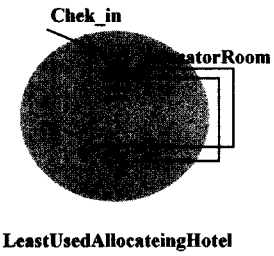

그림 1 상속과 템플릿 메소드(Inheritance and the TempleteMethod) 기법

```
class Hotel{
        Allocator allocator;
        public void checkInGuest(Guest g)
        {    allocator.doAllocation(g);   }
}

interface Allocator{
        Room doAllocation(   ); //return a free room
}

class LeastUsedAllocator implements Allocator{
        Room doAllocation(   ) {   code    }
}

class EvenSpaceAllocator implements Allocator{
        Room doAllocation(   ) {   code    }
}
```

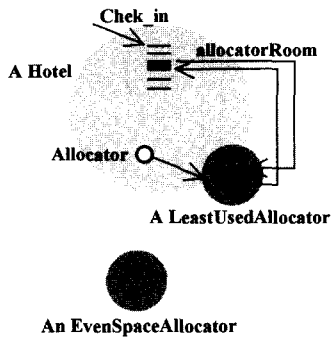

그림 2 다형성과 전달(Polymorphism and Forwarding) 기법

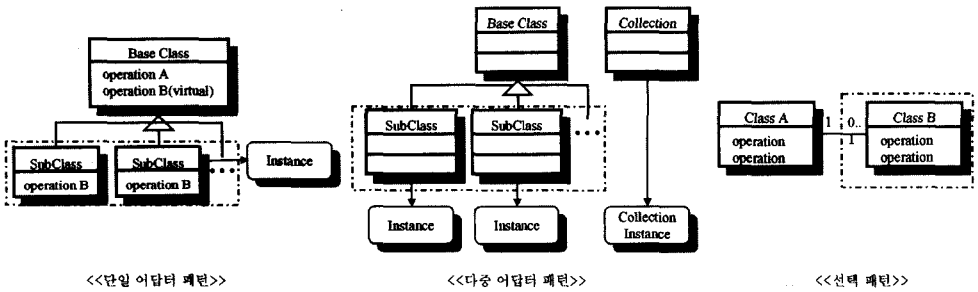

그림 3 Keepence의 가변성 설계 기법

2.2 Keepence의 기법

Keepence는 가변성 설계를 위한 몇 가지 모델을 제시하고 있다. 가변성을 단일 구분(Single Discriminant), 다중 구분(Multiple Discriminant), 그리고 선택(Option)의 3가지 타입으로 분류하고 각각을 위한 모델링 기법을 제시하였다[4].

단일 구분 타입 가변성은 상속을 통하여 그림 3의 좌측 그림과 같은 단일 어답터 패턴으로 모델링 될 수 있다. 공통되는 일반적인 Feature들은 상위 기본 클래스에 두고 하위 클래스에서 상속 받아 개별화 하는 것이다. 이때 하위 클래스는 하나만 인스턴스화 될 수 있도록 하여야 한다. 다중 구분 타입은 가운데 그림과 같이 다중 어답터 패턴으로 모델링 된다. 단일 구분 타입과 같이 상속을 통하여 모델링 되는데, 차이점은 하나 이상의 하위 클래스들이 인스턴스화 될 수 있으며 콜렉션 클래스로 함께 묶이게 된다.

Keepence은 소프트웨어에서 보여지는 가변적 기능의 수적인 측면과 유무 측면에서 가변성 타입들을 구분하였으며, 그를 구현 하기 위한 설계 패턴들을 제시하였다.

2.3 COM 컴포넌트의 가변성 구현 기법

COM 컴포넌트의 다양성 구현 기법에서는 컴포넌트의 가변적인 부분을 식별 및 분류하고, 각각에 맞는 가변성 구현 기법을 제시하고 있다[5]. COM 컴포넌트의 가변성 구현 기법에서는 기능적 측면을 고려한 논리 가변성에 대해서 4가지의 구현 기법을 제시하고 있다. 논리 가변성에서 제시된 4가지 기법은 StaticMethod, Method&ExtComponent, Class&ExtComponent, 그리고 ExtComponent기법이다. 각각의 기법들은 가변성을 가지는 컴포넌트의 내부, 혹은 외부에서 가변 알고리즘을 제공할지에 대한 결정과 가변 알고리즘을 메소드, 클래스, 혹은 컴포넌트 단위로 제공할지에 대한 결정으로 구분 된다. 그림 4에서는 하나의 예로서 Method& ExtComponent기법을 보여주고 있다. 컴포넌트 A가 가지는 임의의 메소드 DoIt()은 컴포넌트에 구현되어 있는 다수개의 메소드(Method 1에서 Method n)로 실제 기능 수행을 전달하게 된다. 각각의 메소드는 DoIt()을 수행하기 위한 서로 다른 알고리즘을 가지고 있으며, 어떤 메소드에 전달할지는 어플리케이션에서 set()메소드를 통해서 파라미터를 전달함으로서 결정한다.

COM 컴포넌트의 가변성 구현 기법에서는 제시된 기법들에 대해 구현 용이성과 확장성, 그리고 가변성 특성에 따른 장단점을 비교, 평가하고 적용 지침도 제공하고 있다.

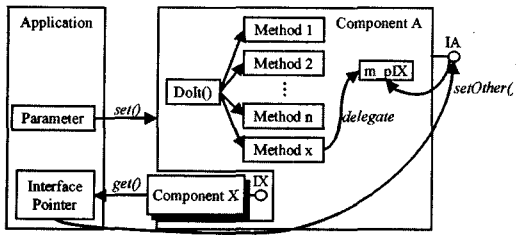

그림 4 논리 가변성을 위한 Method&ExtComponent 기법

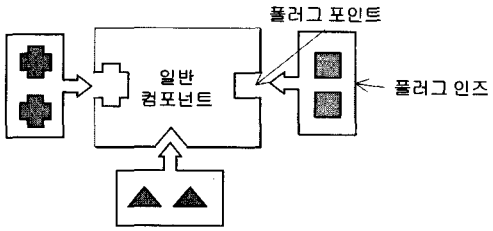

그림 5 Catalysis에서의 일반 컴포넌트와 플러그 포인트, 그리고 플러그 인즈의 관계

3. 기초 연구

3.1 컴포넌트 인터페이스

CBSE[1], PLE[2], CCM[6], Koala[7] 등에서 나오는 여러 컴포넌트 모델에서는 공통적으로 컴포넌트의 인터페이스를 Provide 인터페이스와 Required 인터페이스 두 가지로 분류하고 있다. 각각에서 표현은 달리하고 있지만 Provide 인터페이스는 구현될 컴포넌트가 가지는 기능성을 외부로 보여주기 위한 통로로 정의하고 있다. 즉, 컴포넌트가 가지는 기능성을 나타내는 것이 Provide 인터페이스라고 말하고 있다. 한편 Required 인터페이스는 Provide 인터페이스와 상반되는 개념으로 정의 된다. 컴포넌트 외부에 존재하는 소프트웨어 요소에 대한 의존성을 나타내기 위한 장치를 Required 인터페이스로 정의하고 있다.

3.2 컴포넌트의 가변성

CBD에서는 임의의 컴포넌트가 여러 어플리케이션에서 항상 동일하게 사용되기 보다는 대부분 어플리케이션에 맞게 조정이 되어야 하는데 이를 커스터마이즈라고 한다. 대표적인 CBD 방법론인 Catalysis[3]에서는 산출물들이 여러 상황에서 재사용 될 수 있도록 공통성을 파악하여 일반화한 컴포넌트인 일반(Generic) 컴포넌트를 정의하고, 어플리케이션 개발 시에 발생하는 다양한 요구들을 수용하기 위해서 플러그 포인트를 제공하도록 하고 있다. 플러그 포인트는 가변적 요구사항을 충족시키기 위해 특화된 모듈인 플러그 인즈를 외부로부터 받아들이는 접합점 역할을 하게 된다. 일반 컴포넌트와 플러그 포인트 및 플러그 인즈가 그림 5에서 보여지고 있다.

PLE[2]에서는 어플리케이션들 사이에서 발생하는 차이점들을 가변성 이라 정의한다. PLE에서는 두 가지의 프로세스를 가지고 있는데 프레임워크 프로세스와 어플리케이션 프로세스이다. 프레임워크 공학에서는 여러 패밀리 멤버들과 유사한 어플리케이션을 지원하기 위해 공통성과 가변성을 결정짓기 위한 세부 엑티비티를 정의 한다. 또한 어플리케이션 공학에서는 프레임워크 공학을 통해 개발된 프레임워크를 사용하여 개개의 어플리케이션을 구축하는 프로세스를 가진다

CBD및 PLE와 관련된 여러 논문에서는[8,9] 가변성이 발생되는 지점을 가변점이라 하고 있으며, 가변점은 하나의 가변성에 대해 다수가 존재할 수 있다. 이렇게 변화가 발생하는 가변점은 컴포넌트 소비자의 결정에 의해 특정한 값을 가지게 되는데, 이와 같은 값을 가변치라고 하며, 가변점에 가변치를 바인딩하는 행위를 커스터마이즈라고 한다. 본 논문에서도 이와 같은 용어를 동일한 의미로 사용한다.

3.3 가변성 유형

컴포넌트에서 가변성을 지원하기 위해서는 우선 컴포넌트에 존재하는 가변성을 분류하기 위한 기준이 필요하다. PLE에서 쉽게 볼 수 있는 특성(Feature)에 의한 가변성 분류[10-12]는 컴포넌트 자체가 가지는 가변성에 적용되기에 충분하지 않다. 단지 특성이라는 상위 수준의 개념으로는 컴포넌트가 가지는 특징들을 충분히 반영하지 못하기 때문이다. 컴포넌트가 가지는 가변성은 컴포넌트를 이루고 있는 개념적인 요소들과 밀접한 관계를 가지며 적절한 수준의 유형 분류가 필요하다. 컴포넌트의 가변성은 속성, 논리(알고리즘), 워크플로우, 영속성으로 나누어질 수 있는데[13], 본 논문에서는 컴포넌트가 가지는 기능적 측면의 가변성을 고려하며, 컴포넌트의 기능성을 구성하는 요소를 논리와 워크플로우로 구별한다.

3.3.1 논리 가변성

컴포넌트는 더 작은 단위인 클래스들의 집합으로 이루어 진다. 잘 정의된 클래스들은 단일 기능을 처리하기 위한 메소드들을 가지고 있다. 논리 가변성은 이렇게 클래스가 가지는 단일 메소드 상에서 기능을 수행하기 위한 알고리즘이 다양할 경우에 식별된다[13]. 논리 가변성은 단일 메소드 내부에서 알고리즘 상의 차이만을 말하는 것이며 다른 클래스와의 관계성, 예를 들어 메시지 의존성 같은 측면에서는 동일한 경우이어야 한다.

3.3.2 워크플로우 가변성

컴포넌트의 워크플로우는 여러 개의 메시지로 이루어

지는데, 동일한 기능을 수행하기 위해 다양한 메시지의 흐름이 존재 할 수 있다. 워크플로우 가변성은 기능 수행을 위한 객체들간의 메시지 흐름이 가변적일 경우에 나타난다[13]. 워크플로우는 메시지의 흐름에 대한 제어가 집중되는 형태와 분산되는 형태로 나타날 수 있다.

3.4 가변성 범위(Scope)

가변적인 부분은 프레임 공학에서 볼 수 있는 것처럼 이미 고려된 여러 도메인에서 파악될 수 있는 가변성과 새로운 어플리케이션을 개발할 때 식별되는 새로운 가변성이 있다. 컴포넌트는 이상적으로 블랙 박스를 지향하고 있으며, 내부 수정이 힘들다[14]. 따라서 컴포넌트의 가변성을 분류할 때는 식별된 가변성과 식별되지 않은 가변성에 대한 고려가 있어야 한다. 이는 컴포넌트가 지원하는 가변치들에 대한 확장성과 연관을 가지며, 가변성의 범위는 가변성이 가지는 가변치들의 범위를 의미한다.

3.4.1 닫힌 범위(Closed Scope)

닫힌 범위의 가변성은 임의의 가변적 기능에 대한 가변 범위가 이미 식별 되어 있음을 나타낸다. 즉, 모든 가변치들이 식별된 경우이다. 식별된 가변치들은 컴포넌트 개발 시에 미리 반영하여 구매자가 구현된 기능을 이용할 수 있도록 할 수 있다. 이는 PLE에서의 프레임 워크 공학 동안에 여러 어플리케이션들 사이에서 발생되는 가변치들이 모두 식별 된 경우이다.

3.4.2 열린 범위(Open Scope)

열린 범위의 가변성은 가변적인 값의 범위가 일부만이 식별이 된 경우이다. 즉, 가변성에 대한 가변치들이 변경될 수 있는 경우이다. 동일 도메인에 존재하는 여러 어플리케이션 상에서 특정 기능성이 요구되어 기능 자체는 식별이 되었지만, 구체적으로 기능에 사용할 알고리즘이나 데이타 등에 대한 정의가 아직 미흡할 때 새로 추가되는 기능적 요구 사항 등에 대해 나타날 수 있다.

4. 가변성 설계 기법

4.1 선택형(Selection) 기법

4.1.1 논리 가변성 선택형 기법

논리 가변성의 선택형 기법은 3장에서 정의되었던 닫힌 범위의 논리 가변성을 구현하기 위한 기법으로서 컴포넌트 내부에 가변적인 경우의 수를 포함하여 컴포넌트 구매자에 의해 선택적으로 컴포넌트를 커스터마이즈 할 수 있도록 해주는 기법이다.

컴포넌트의 기능 중에 특정 메소드를 수행하는데 있어서 도메인 내의 여러 멤버들 간에 다른 알고리즘들을 가지고 있다면, 각각의 멤버들을 위한 다양한 알고리즘을 제공해야 한다. 이러한 가변적 성향의 기능을 가진 메소드는 논리 가변성에서 가변점으로 식별 된다.

컴포넌트를 위한 선택형 기법은 그림 6과 같은 구조를 가진다. 좌측에 보이는 컴포넌트 클라이언트는 다른 컴포넌트 이거나 클라이언트 프로그램이 될 수 있으며, 컴포넌트 커스터마이저는 컴포넌트를 커스터마이즈하기 위한 가벼운 프로그램이거나 툴일 수 있다. 가변적인 알고리즘이 클래스 A내부의 opA로 반영이 되어 있을 때, opA는 다양한 알고리즘을 가지는 가변점이 되는데, opA는 opA_1과 opA_2의 두 가지의 가변치를 가지고 있음을 볼 수 있다. 이러한 가변치들과 가변점, 그리고 이들을 바인딩하기 위한 바인딩 변수 cvOpA(Current Variant of opA)는 모두 가변점과 동일한 클래스에서 구현된다. 컴포넌트가 opA_2의 알고리즘을 이용하는 어

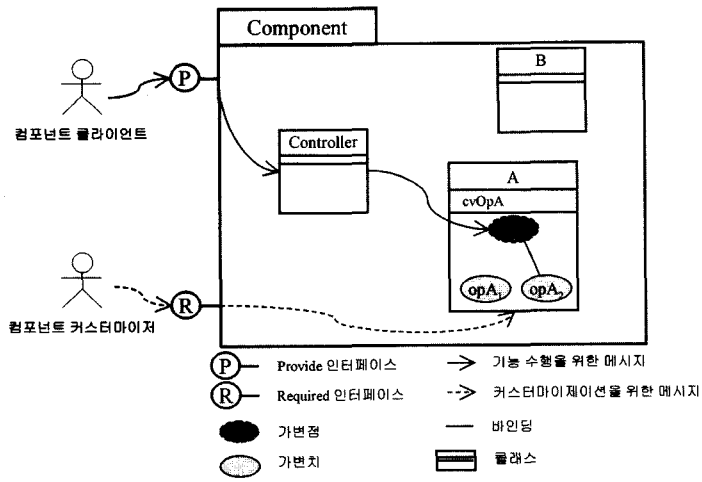

그림 6 논리 가변성 선택형 기법의 개념적 구조

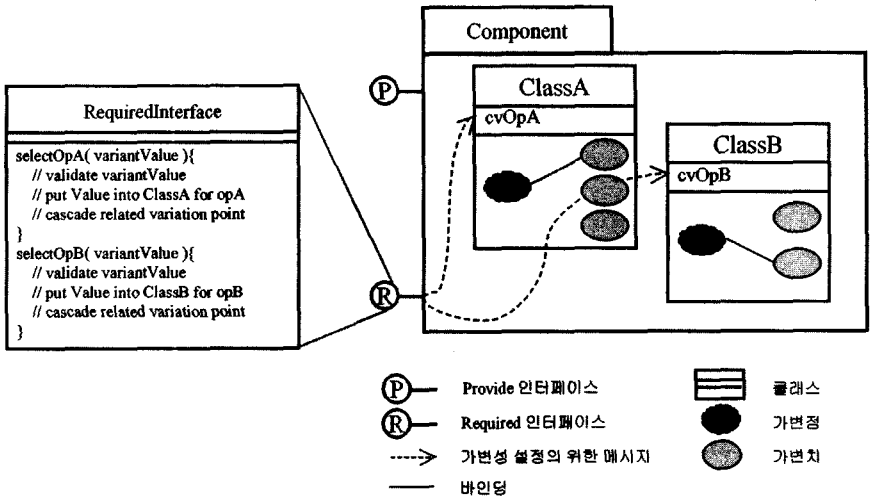

그림 7 가변성 커스터마이즈를 위한 Required 인터페이스

플리케이션에서 사용될 경우, 컴포넌트 커스터마이저가 Required 인터페이스를 통해 가변치 opA_2에 해당하는 선택값을 클래스 A의 정적 변수 cvOpA에 할당하게 되면 가변점인 opA()는 가변치 opA_2에 바인딩 된다. 일단 가변성 설정이 완료되면 알고리즘 opA를 수행할 때 바인딩 변수를 참고하여 opA_2로 기능을 전달하게 된다. 앞에서 언급 했듯이 논리 가변성에서는 가변점과 가변치, 그리고 바인딩 변수가 같은 클래스에 존재하게 되는데 바인딩 변수의 경우 클래스의 모든 인스턴스에서 동일하게 적용이 되어야 하기 때문에 정적(Static) 타입의 속성을 가진다.

그림 7은 가변성을 커스터마이즈 하기 위한 Required 인터페이스를 보여주고 있다. Required 인터페이스는 컴포넌트의 가변성을 설정해 주기 위한 장치로서, 컴포넌트 내부에 존재하는 가변점과 가변 범위에 대한 정보를 가지고 있게 된다. 그림에서는 Required 인터페이스 내부에 두 개의 오퍼레이션이 있는데 이것은 컴포넌트 내부에 두 개의 논리 가변성 가변점이 존재함을 나타낸다. 또한 오퍼레이션의 접두사가 select로 시작하는 것은 선택형 기법이 적용된 가변성임을 나타낸다.

Required 인터페이스의 오퍼레이션을 구현한 메소드는 우선 입력된 가변성 설정값에 대한 검증을 해야 하며, 설정값이 가변 범위에 적합한 경우에만 가변점에 적용이 될 수 있도록 해당 클래스에 저장해야 한다. 또한 가변성 설정 메소드는 논리 가변성간의 연관 관계에 대한 정보도 가지고 있어야 하며, 하나의 논리 가변성과 연관된 다른 논리 가변성이 있다면 일관성 있는 설정을 위해 다른 가변점도 일관성 있게 설정할 책임을 가진다.

구현 시에 가변점이 되는 opA()에서는 기능 수행을 위한 알고리즘이 아니라 바인딩 변수를 참고하여 올바른 가변치 메소드를 호출해 주는 알고리즘만이 존재한다. 따라서 IF문이나 SWITCH와 같은 비교, 선택문을 통해 구현해 줄 수 있다. 이 방법은 구현이 간편하지만, 매우 빈번히 수행되는 알고리즘일 경우, 수행 시 매번 바인딩 변수를 참고하여 알고리즘을 선택해 주기 때문에 수행성에 좋지 않다. 따라서 이런 경우에는 하나의 클래스에 하나의 가변치 메소드를 구현하여 바인딩 변수에 선택값이 아니라 가변치 알고리즘을 가지는 클래스의 객체를 저장하면 성능을 개선할 수 있다.

논리 가변성을 위해 이와 같이 컴포넌트를 설계하게 되면 컴포넌트를 블랙 박스 형태를 유지하면서도 다양한 알고리즘을 수행할 수 있도록 할 수 있고, 개발이 용이해지며, 컴포넌트 소비자가 컴포넌트를 쉽게 커스터마이즈 할 수 있다.

4.1.2 워크플로우 가변성 선택형 기법

워크플로우 가변성의 선택형 기법은 닫힌 범위의 워크플로우 가변성을 구현하기 위한 기법이다. 도메인 내의 여러 멤버들이 원하는 특정 기능에 대해 다양한 워크플로우를 컴포넌트 내부에 구현하여, 컴포넌트 구매자들이 자신들의 시스템에 맞도록 선택적으로 컴포넌트의 워크플로우를 커스터마이즈 할 수 있도록 한다.

그림 8에서는 컴포넌트 내부에서 특정 기능을 수행하기 위해 두 개의 워크플로우가 존재함을 보여주고 있다. 기능 W를 제공하기 위해 컴포넌트의 워크플로우를 수행하는 컨트롤러 클래스가 외부로부터 Provide 인터페이스를 통해 서비스 요청을 받아 기능 수행에 필요한 워크플로우를 진행 시키게 된다. 기능 W를 위한 컨트롤러 클래스의 메소드를 opW라고 하면, opW는 A→B→C

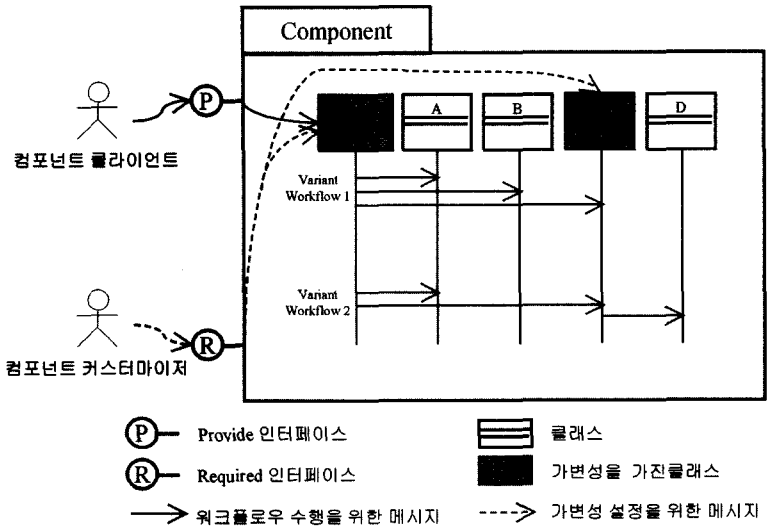

그림 8 컴포넌트의 워크플로우 가변성

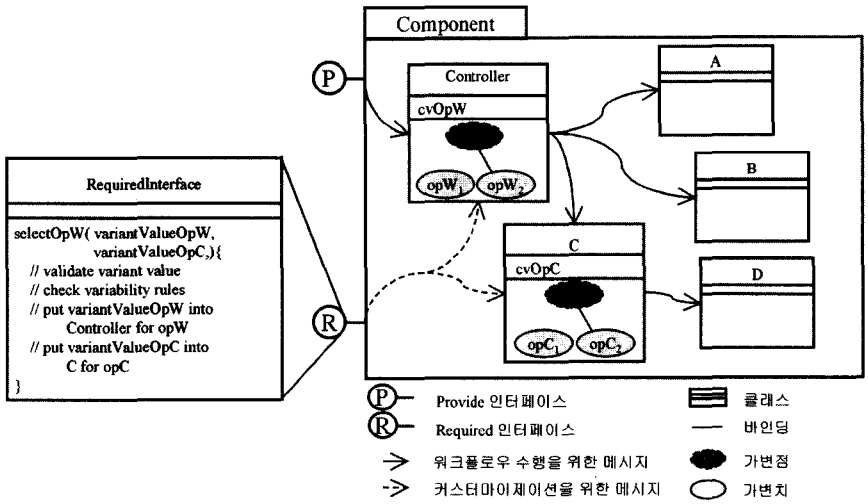

그림 9 워크플로우 가변성을 위한 선택형 기법의 구조

와 A--C--D의 두 가지 워크플로우를 가지고 있다. 따라서 워크플로우 W를 위한 가변점은 컨트롤러의 메소드인 opW()가 된다. 또한 분석된 컴포넌트의 워크플로우를 살펴보면 클래스 C에서도 워크플로우 유형에 따라 클래스 D로의 메시지를 가변적으로 생성시키고 있음을 알 수 있다. 즉, 워크플로우 1의 경우는 C에서 D로의 메시지가 없으며, 워크플로우 2의 경우에는 메시지가 발생하고 있다. 따라서 클래스 C의 메소드인 opC()역시 워크플로우 W의 가변점이 된다. 즉, 3장 에서 살펴 보았듯이 하나의 워크플로우 가변성에 대해 두 개의 가변점이 존재하는 경우이다.

그림 9에서는 그림 8에서 나타난 컴포넌트의 워크플로우 가변성을 위한 선택형 기법의 구조와 Required 인터페이스를 보여주고 있다. 컨트롤러 클래스와 클래스 C의 오퍼레이션 opW()와 opC()는 그림 8에서 보여지는 워크플로우1과 워크플로우 2를 위한 두 가지의 가변적 메시지 흐름을 가지게 된다. 각각 opW1 과 opW2, 그리고 opC1과 opC2를 가변치 메소드로 가지고 있으며, 가변점과 가변치간의 바인딩은 각각의 해당 클래스 내에 가변성 설정 정보를 저장하기 위한 정적 속성값을 통해서 하게 된다. 논리 가변성과 유사하지만, 중요한 차이점은 대상이 되는 가변적 메소드가 클래스 외부로

메시지 흐름을 발생시키는지의 유무에 가변성이 존재하고 있다는 것이다. 또한 Required 인터페이스에서는 그림에서 볼 수 있듯이 하나 이상 존재하는 가변점에 대한 설정값들을 매개 변수로 받아 각각의 바인딩 변수에 설정해 주어야 한다. 또한 설정되는 가변치에 대한 범위뿐만이 아니라, 동시에 설정될 수 없는 가변치들에 대한 설정을 피하기 위해 가변치들간에 존재하는 규칙도 검사하여야 한다.

논리 가변성과 마찬가지로 워크플로우의 가변치들은 가변점이 되는 동일 클래스에 존재하게 되는데 이때 간단한 워크플로우인 경우 동일 메소드에서 구현해 줄 수도 있지만 가독성과 유지 보수성을 위해 다른 메소드들로 따로 관리해 주는 것이 좋다. 워크플로우 가변성을 위해 이와 같이 컴포넌트의 구조를 구성하게 되면 컴포넌트를 블랙 박스 형태를 유지하면서도 다양한 워크플로우를 수행할 수 있도록 할 수 있다. 이렇게 컴포넌트 내부에 미리 구현된 워크플로우를 선택하는 워크플로우 선택형 기법은 컴포넌트 구매자에게 쉽고 안정성이 높은 커스터마이즈 방법을 제공하게 된다.

4.2 플러그인(Plug in) 기법

4.2.1 논리 가변성 플러그인 기법

논리 가변성 플러그인 기법은 열린 범위를 가지는 논리 가변성을 구현하기 위한 기법이다. 즉, 동일 도메인 상에서 임의의 기능성에 대해 필요성은 인식되어 있지만, 멤버들 간에 구현하는 내부 알고리즘이 매우 다양할 경우, 혹은 미래에 확장 가능성이 있는 기능일 경우에는 미리 컴포넌트 내부에 구현하기가 힘들다. 따라서 컴포넌트 내부에는 기능성에 대한 선언만을 해놓고 외부에서 기능을 정의하여 구현한 실제 알고리즘을 넣어줄 수 있도록 해주는 것이 플러그인 기법이다. 이를 위해서는 컴포넌트 내부에 미리 플러그인을 위한 설계가 이루어져야 한다.

그림 10은 논리 가변성 플러그인을 위한 설계를 보여준다. 논리 가변성이 존재하는 가변점인 opA는 추상 클래스 AbsPlugins로 기능 수행을 전달한다. 추상 클래스 AbsPlugins에는 가변점이 되는 오퍼레이션과 동일한 오퍼레이션이 선언되어 있다. 즉, opA 내부에는 기능 수행을 위한 어떤 알고리즘도 없으며 구체적인 기능은 클래스 AbsPlugins의 오퍼레이션으로 전달하게 된다. 클래스 AbsPlugins는 추상 클래스로서 실제 기능을 수행할 객체는 외부에서 추상 클래스 AbsPlugins를 상속받아 구현하며, 구현된 구체 클래스 Plugins의 객체 plugins를 Required 인터페이스를 통해 컴포넌트에 전달한다. Required 인터페이스는 가변치 객체를 사용할 가변점에 대한 정보를 가지고 있으므로 가변점을 가지는 클래스 A의 바인딩 변수인 cvOpA에 플러그인 된 plugins를 저장한다. 따라서 Provide 인터페이스를 통해서 클래스 A의 opA로 메시지 호출이 오면 opA에서는 저장된 플러그인 객체를 사용하여 알고리즘을 수행하게 된다.

이때 플러그인 된 객체는 추상 클래스인 클래스 AbsPlugins를 상속 받은 클래스의 객체이므로 추상 클래스 AbsPlugins를 인터페이스로 하여 항상 동일하게 사용할 수 있다. 즉, 논리 가변성 플러그인 기법은 외부 객체가 플러그 될 지점을 추상 클래스로 두고 추상 클래스를 상속한 다양한 가변치 클래스의 객체를 컴포넌트에 플러그인 함으로서 여러 멤버들이 각각의 시스템의 목적에 맞는 알고리즘을 컴포넌트에 넣어 줄 수 있도록 한다. 그림 11은 가변점과 가변치가 되는 메소드와 바인딩 변수, 그리고 바인딩 변수의 설정 함수를 보여준다. 바인

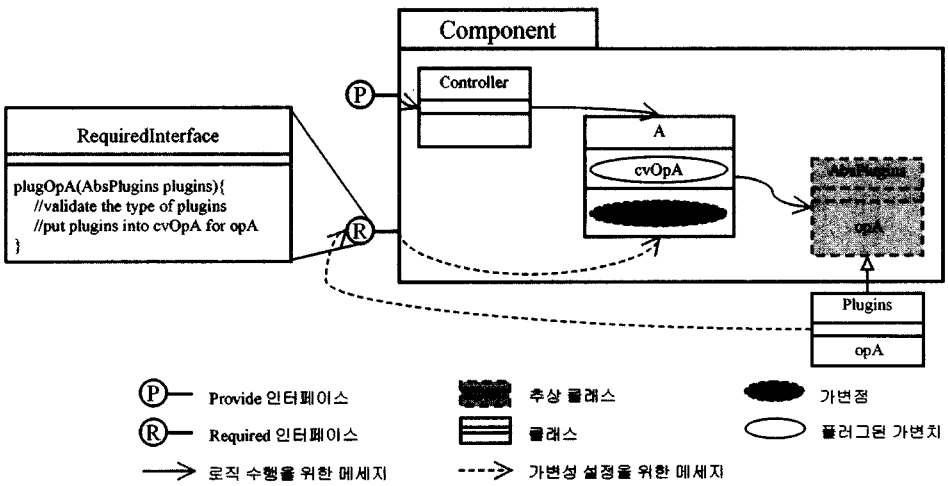

그림 10 논리 가변성 플러그인 기법의 구조

```
// In A class
AbsPlugins cvOpA;        // binding varianble stores the current-variant object

public void opA(){       // variation point whose current-variant variable is object
    // perfrom cvOpA.opA()
}
public static void setOpA(AbsPlugins plugins){      // set method of current-variant variable
    // put plugins into variable cvOpA
    // call saveCvOpA(plugins)
}
public static void saveCvOpA(AbsPlugins plugins){
    // store plugins in persistent storage
}
public static void restoreCvOpA(){                  // restore the configured variability
    // get plugins from persistent storage, and call setCvOpA(plugins) method
```

```
// In Plugins class
class Plugins implements java.io.serializable{          // variant object
    public void opA(){                                  // variant method
        // logic for function A
    }
}
```

그림 11 논리 가변성 플러그인 기법에서 가변점과 가변치가 되는 메소드 예제

딩 변수인 cvOpA는 플러그인 될 객체의 상위 클래스인 AbsPlugins 클래스 타입이며 가변점 메소드인 opA에서는 바인딩된 객체의 메소드로 기능을 전달하고 있다. 또한 설정된 가변성의 영속성을 위해서 가변치가 되는 객체는 저장될 수 있어야 하며, 이를 위해 예제에서 가변치 클래스는 직렬화 되기 위하여 java.io. serializable 인터페이스를 구현하고 있는 것을 볼 수 있다. 선택형 기법에서와 같이 setter 함수인 setCvOpA는 가변치 객체를 바인딩 변수에 할당하고, 영속적인 저장소에 저장하기 위해 saveCvOpA() 메소드를 호출한다. 또한 이전에 설정된 가변치 객체를 복구하기 위해 restore-CvOpA()메소드를 정의하고 있다. 가변성의 영속적인 설정에 대한 부분은 선택형 기법에서 .NET과 EJB를 예로 들어 제시했던 방법으로 동일하게 해결 된다.

한편 Required 인터페이스에서는 선택형 기법과 다르게 가변 범위에 대한 검증이 아니라 플러그인 되는 객체의 타입 검증을 하고 그에 대한 에러 처리를 수행 한다. 플러그인 기법에서는 가변치가 컴포넌트 내부에 존재하지 않으며, 커스터마이즈 오퍼레이션의 매개 변수가 가변치 클래스의 객체인 것이 선택형 기법과의 차이점이다. 플러그인 기법에서의 가변점과 정적 바인딩 변수, 그리고 Required 인터페이스의 커스터마이즈 오퍼레이션은 다음과 같이 표기한다.

- Current Variant
  := cv<VariationMethodName>
- Variation Point
  := <VariationMethodName>()
- Customizing Operation
  := plug<VariationMethodName>(variantObject)

논리 가변성은 단순히 가변점의 알고리즘뿐만이 아니라 처리하는 데이타의 차이에 의해서도 발생할 수 있다. 제시된 논리 가변성 플러그인 기법은 데이타의 다양성에 의한 논리 가변성에 대해서도 효과적인 해법을 제공한다. 그림 12는 다양한 데이타를 처리하는 알고리즘을 플러그인 하기 위한 설계이다. 컴포넌트 내부의 AbsDO라는 추상 클래스는 데이타 객체(Data Object 혹은 Value Object)를 위한 추상 데이타 클래스 역할을 한다. 그림 10과 다른 점은 AbsDO라는 추상 데이타 클래스가 추가 되었으며 가변점과 컴포넌트 내부에 존재하는 추상 가변치가 모두 추상 데이타 클래스를 사용하고 있다는 것이다. 또한 실제 플러그인 될 객체의 클래스는 추상 가변치를 구현하고 있는 것뿐만 아니라 추상 데이타 클래스를 상속 받은 가변치 데이타 클래스를 사용하여야 한다. 추상 클래스는 내부가 비어 있는데 하위 데이타 클래스들을 모두 동일한 변수로 다루기 위한 장치가 된다. 이를 통해 외부와 연결되는 컴포넌트의 논리적

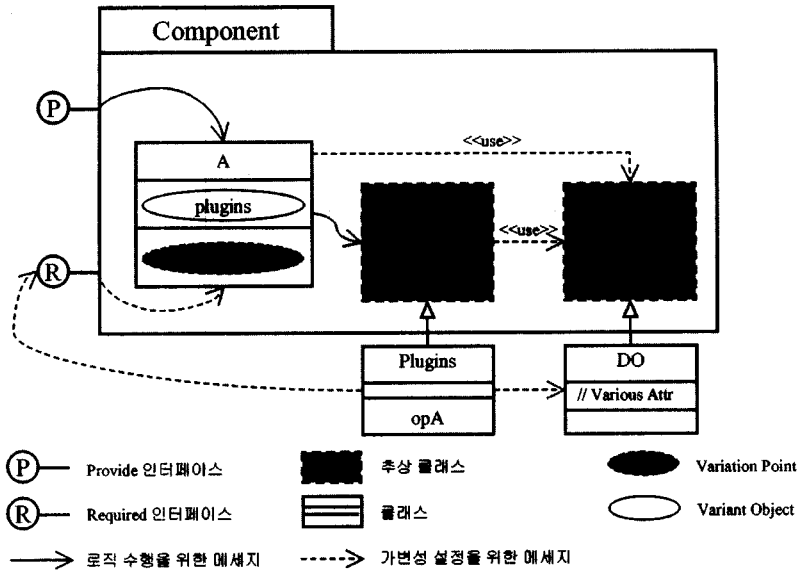

그림 12 데이타 다양성에 의한 논리 가변성

외곽 지점에서 나타나는 파라미터상의 가변성을 처리함으로서 컴포넌트간의 연결성을 높여 컴포넌트 통합 시 적응성을 높일 수 있다.

논리 가변성을 위한 플러그인 기법은 선택형 기법에서의 커스터마이즈 방법과 다르게 가변치에 대한 선택값이 아니라 가변 알고리즘을 수행하기 위한 가변치 객체 자체가 바인딩 변수에 저장된다. 논리 가변성 플러그인 기법을 통해서 다양한 가변 알고리즘들에 대해 유연성 있고 확장성 있는 논리 가변성의 구현이 가능하다. 그러나 가변치 객체를 컴포넌트에 플러그인 하기 위해서는 추상 클래스의 알고리즘을 구현해야 하기 때문에 컴포넌트 구매자가 직접 원하는 알고리즘을 구현하여야 하며, 플러그인 된 알고리즘에 대한 테스트도 직접 해주어야 한다는 점에서 더 많은 노력이 요구된다. 또한 선택형 기법에서는 연관되는 가변점에 대한 관리를 컴포넌트에서 해 주었기 때문에 가변성간의 일관성을 유지하는데 용이 했지만, 플러그인 기법에서는 가변성간의 일관성 관리에 대한 책임이 컴포넌트 커스터마이저에게 있게 된다.

4.2.2 워크플로우 가변성 플러그인 기법

워크플로우 가변성을 위한 플러그인 기법은 컴포넌트가 가지는 열린 범위의 워크플로우 가변성을 위한 기법이다. 즉, 컴포넌트에서 워크플로우를 구성하는 메시지들간의 흐름정보를 외부로부터 객체 형태로 컴포넌트에 넣어줌으로써 멤버들이 가지는 다양한 워크플로우를 제공해 주기 위한 기법이다. 워크플로우를 구현한 가변치를 생성하기 위해서는 우선 워크플로우를 구성하는 컴포넌트 내부의 함수들이 식별 되어야 한다.

표 1은 가변적인 워크플로우에 대한 명세를 제공하는 테이블이다. 가변적인 워크플로우에 참여하는 함수 즉, 메시지들은 C&V 분석 단계에서 이미 식별되어 있음을 가정한다. 식별된 메시지들은 객체나 컴포넌트 상에 존재하는 메시지들을 대상으로 하며, 각각의 메시지를 이루는 메소드와 입출력 파라미터 등이 함께 제공되어야 한다. 이러한 메시지들의 흐름을 관리하기 위한 지점은 워크플로우 가변성의 가변점으로 식별이 된다.

가변성 워크플로우 명세 테이블은 컴포넌트 구매자가 컴포넌트의 워크플로우 가변성을 커스터마이즈 하기 위한 기초 자료로서 사용된다. 테이블은 워크플로우 가변성 이름, 커스터마이즈 메소드, 플러그인과 객체 접근

표 1 가변적 워크플로우 명세 테이블

Variability Name	Customize Method		AbsPlugins		Object Access Method	Clips		
	Required Interface	Operation	Class	Method		Object	Method	IOParam
WV_1	IR_1	OP_1	PC_1	PM_1	OAM_1	CC_1	CM_1	Id_1, Od_1

메소드, 그리고 워크플로우를 구성하는 클립(Clip)들을 명시하고 있다.

커스터마이즈 메소드(Customize Method)는 가변 워크플로우의 흐름 정보를 컴포넌트가 플러그인 객체를 받기 위한 통로 역할을 하는 Required 인터페이스와 오퍼레이션을 명시한다. 추상 플러그인(AbsPlugins)은 컴포넌트 내부에 플러그인 될 객체의 인터페이스를 명시하는데 다양한 가변치들간의 공통 인터페이스가 되는 추상 클래스가 되며 구현해야 할 추상 메소드를 가진다. 플러그인 될 가변치 객체는 명시된 추상 클래스를 구현한 클래스의 인스턴스가 된다. 객체 접근 메소드(Object Access Method)는 클립 메소드들을 가지는 객체들을 획득하게 해주는 메소드이다. 마지막 항목인 클립은 컴포넌트 내부에서 워크플로우에 참여하는 메시지들의 조각을 의미하는데, 각각 워크플로우를 구성하는 컴포넌트 내부의 메시지들에 대한 정보를 명시해준다. 테이블을 기반으로 하여 작성된 클래스는 가변성 워크플로우의 가변치 메소드를 구현하게 되는데, 이를 객체화하여 컴포넌트에 플러그인 하게 된다.

그림 13은 워크플로우 가변성을 지원하기 위한 플러그인 기법을 컴포넌트 내부에 설계한 그림이다. AbsPlugins라는 추상 클래스는 외부로부터 플러그인 될 객체에 대한 인터페이스를 제공한다. 플러그인 될 Plugins 클래스에서 실제 워크플로우를 구현하는 것은 논리 가변성과 동일하다. 그러나 AbsPlugins는 ClipAgent라는 클래스를 상속하고 있으며 ClipAgent는 워크플로우의 클립 메소드를 가지는 객체들에게 접근할 수 있는 방법을 제공한다. ClipAgent 클래스는 내부에 워크플로우에 참여하는 객체들을 속성으로 가지며, 이들 속성을 생성하거나 찾아오기 위한 메소드를 가진다. 그림에서 볼 수 있는 단일 객체들뿐만이 아니라 필요에 따라 유연하게 객체들의 컬렉션(Collection)들도 속성으로 정의하여 사용할 수 있다. 이 속성과 메소드들은 모두 Protected 타입의 가시성을 가지기 때문에 자식 클래스인 Plugins밖에는 접근할 수 없으며, 속성이 되는 객체들에는 직접적인 접근이 가능하게 된다. 따라서 Plugins는 객체 접근 메소드를 통해서 획득된 컴포넌트 내부의 객체들에게 직접 메시지 전달이 가능하게 된다. Plugins를 작성하는 컴포넌트 구매자는 컴포넌트 내부의 클래스 A, B, C에 대해서는 알지 못하고, 단지 워크플로우에 참여하는 클립들의 메소드와 메소드를 실행할 객체들만을 알고 있게 되며 따라서 블랙 박스 형태의 컴포넌트 상에서의 워크플로우 플러그인이 가능하게 된다.

그 외에 추상 클래스에 존재하는 가변치 메소드를 자식 클래스인 Plugins에서 구현하는 것과, 구현된 Plugins 클래스가 객체화 되어 가변점인 Controller의 바인딩 변수에 Required 인터페이스를 통해서 저장 되는 것은 논리 가변성의 플러그인 기법과 동일하다. 또한 워크플로우 선택형 기법에서 보았듯이 가변점은 하나 이상일 수 있으며, 따라서 Required 인터페이스의 오퍼레이션은 하나의 워크플로우 가변성에 존재하는 다수의 가변점에 객체를 바인딩하기 위해 가변점과 동일한 수의 가변치 객체를 외부로부터 매개 변수로 받아야 한다. 이것은 워크플로우 가변성을 위한 선택형 가변성에서 보았던 것과 유사한 것이며, 차이점은 변수로 받는 것이 가변치를 위한 선택값이 아니라 플러그인 할 가변치 객

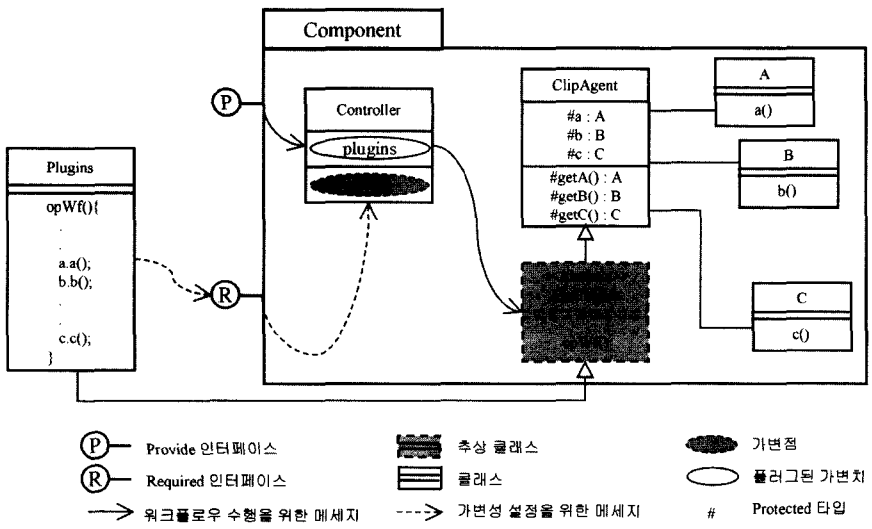

그림 13 워크플로우 가변성을 위한 플러그인 기법의 개념적 구조

체 자체라는 것이다.

워크플로우 가변성을 위한 플러그인 기법은 컴포넌트 내부에 다양한 메시지 흐름을 생성해 줄 수 있기 때문에 컴포넌트에 높은 유연성과 적응성을 제공해준다. 제시된 가변적 워크플로우 명세 테이블의 범위 내에서 원하는 메시지의 추가나 삭제 변경 등이 가능하여 컴포넌트 내부에 없는 기능들은 외부의 컴포넌트를 호출함으로써 워크플로우에 기능을 추가할 수 있다. 이와 같이 설계된 컴포넌트는 구매자가 내부의 워크플로우를 유연하게 확장하면서도 블랙 박스 형태를 가지게 된다.

4.3 외부화(Externalization) 기법

컴포넌트 가변성을 위한 외부화 기법은 컴포넌트 내부에 있는 가변성을 설정하기 위한 정보를 컴포넌트 외부에 두고, 컴포넌트 커스터마이즈 시간에 외부 설정 정보를 참고하여 컴포넌트를 커스터마이즈 하는 것이다. 열린 범위의 가변성에 적합한 기법이다. 간단한 예로 외부화된 정보를 통하여 컴포넌트 내부의 고정 변수에 값을 가변적으로 설정하기 위한 방법으로서 Enterprise JavaBeans(EJB)[15]의 디플로이먼트 디스크립터(Deployment Descriptor)의 구성 요소인 <env-entry>태그 안에서 <env-entry-name>, <env-entry-type>, <env-entry-value>태그를 작성하여 Runtime에 생성되는 빈들이 가지는 속성에 설정값을 할당해주는 것이다.

EJB에서의 배치 관리자가 주로 빈들이 가지는 속성과 관계성에 관련된 명세서인 반면, 본 논문에서 제시하는 가변성 프로파일은 컴포넌트의 가변성 정보를 외부화시켜 놓은 파일이다. 컴포넌트가 가지는 가변성에 참여하는 여러 내부 요소들에 대해 별칭(Alias)형식의 정보를 주어 블랙 박스 컴포넌트를 유지 하면서도 컴포넌트가 가지는 가변성에 대한 정보를 소비자에게 전달할 수 있다. 주어진 정보를 통해서 가변성 프로파일에 명세를 해주게 되면 컴포넌트는 그 정보를 통해서 가변적 부분을 적합하게 커스터마이즈 될 수 있다.

- 가변성 이름과 개요, 가변성 타입
- 가변성에 영향을 받는 가변점 정보
- 가변점에 대한 가변치 정보
- 가변성 규칙 정보
- 가변성 별칭 정보
- 가변성 설정 정보

위의 항목들은 가변성을 외부화 하는데 고려해야 하는 사항들이다. 가장 마지막 항목인 가변성 설정 정보는 컴포넌트 구매자가 커스터마이즈 하기 위해 작성하는 정보이다. 그 이외에는 모두 가변성을 명세하기 위한 항목으로서 고객에게는 가변성 이름과 개요, 타입, 그리고 별칭화 된 정보가 주어진다. 명세된 가변성 정보는 프로파일 처리자에게 포함되어 프로파일을 검증하고 그에 따라 컴포넌트를 커스터마이즈하기 위한 기초 정보로 사용된다. 그림 14에는 고객에 의해 작성된 프로파일과 컴포넌트 개발자에 의해 작성된 가변성 명세 정보를 이용하여 컴포넌트를 커스터마이즈 하는 절차를 보여주고 있다. 프로파일 처리자는 가변성 명세 정보를 바탕으로 프로파일 정보를 분석하여 컴포넌트가 가지는 가변성을 고객의 의도에 맞도록 커스터마이즈 한다.

그림 14에서 보이는 프로파일 처리자는 4 가지 기능을 수행한다. 우선 프로파일 분석을 통해 작성된 프로파일의 유효성을 검사하게 된다. 이때 프로파일 처리자가 가지고 있는 가변성 명세 정보에 맞게 프로파일이 작성되어 있는지를 검사한다. 프로파일이 가변성 명세에 어긋나지 않게 작성이 되었으면, 프로파일에 설정된 가변성 정보를 바탕으로 컴포넌트에 생성할 객체를 위한 소스 코드를 생성하게 되고, 생성된 소스 코드는 컴파일 되면서 다시 한번 유효성 검사를 수행하게 된다. 컴파일

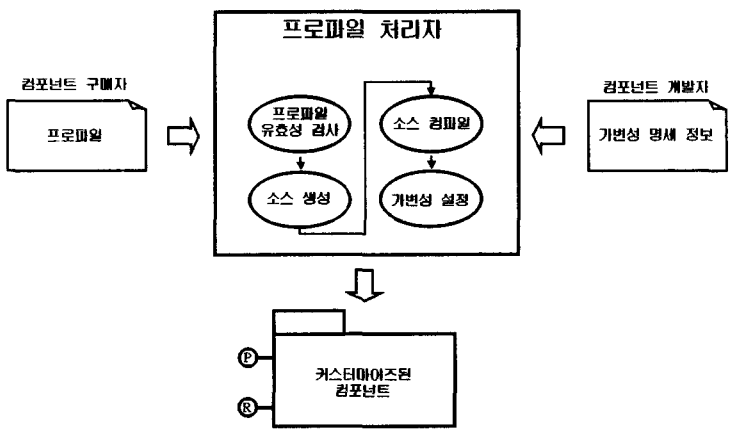

그림 14 프로파일을 통한 커스터마이즈 절차

```
// In the class having variation point
AbstractVariant cvOpA;                          // binding varianble stores the current-variant object

public void variationMethod(){                  // variation point whose current-variant variable is object
    // perform cvOpA.variantMethod()
}
public static void setVariationMethod(AbstractVariant vObj){    // set method of current-variant variable
    // put vObj into variable cvOpA
    // call saveCvOpA(vObj)
}
public static void saveCvOpA(AbstractVariant vObj){
    // store vObj in persistent storage
}
public static void restoreCvOpA(){              // restore the configured variability
    // get vObj from persistent storage, and call setCvOpA(vObj) method
```

```
// In the generated Class
public void variantMethod(){                    // variant method
    // customized code for performing the specific logic
}
```

그림 15 외부화 기법에서 가변점과 가변치의 구현

```
<variability>
  <workflow name=" ">
    <variation-point name=  >
      <init>
        <arg name="arg1" bindingTo="input1  type= String >
        <arg name="arg2" bindingTo="input2  type= int >
      </init>

      <customer-clip name="mb" type= EJB  target="comp1" jndi="jndi1" location="location" bindingOp="opB >
        <arg name="b1" type="String">
        <arg name="b2" type="int">
        <return name="mb_return" type="String">
        <exception name= opBException ></exception>
      </customer-clip>

      <sequence>
        ...
        <switch value=input1>
                <case condition="condition1" do=mb>
                </case>
                ...
        </switch>
        ...

        <while target=m3 value=return < 100">
        </while>
        ...
      </sequence>
      <return-value name=mf_return></return-value>
    </variation-point>
    .
    .
  </workflow>
</variability>
```

그림 16 외부화 기법에서의 워크플로우 프로파일

이 올바르게 수행되었다면 생성된 클래스는 마지막으로 객체화 되어 가변점에 바인딩 된다. 이렇게 설정된 가변성은 플러그인 기법에서와 같이 가변성 설정의 영속성을 유지시킬 수 있다. 그림 15에서 위의 그림은 검증된 가변치 소스가 객체화 되어 가변점에 바인딩 될 때의 구현 소스이며, 플러그인 기법에서와 유사함을 알 수 있다. 아래 그림은 프로파일을 통해서 생성된 가변치 클래스이며 내부에 커스터마이즈를 위한 가변치 메소드를 볼 수 있다. 이렇게 커스터마이즈 된 가변치 클래스를 객체화 하며 가변점을 가지는 클래스의 바인딩 변수에

할당하게 된다.

외부화 기법에서는 프로파일과 가변성 명세 정보를 어떻게 구성하고 작성하느냐가 매우 중요하다. 그림 16에서 워크플로우 가변성 설정을 위해 작성된 프로파일에서 볼 수 있듯이 XML을 이용하여 가변성 설정에 대한 모든 정보를 문서화 해주는 것이다. 가변점, 가변치 그리고 클립에 대한 정의는 플러그인 기법에서와 같다. 외부화 기법에서는 논리 가변성의 경우, 워크플로우 가변성과 동일하게 알고리즘을 프로파일로 작성하게 되며 내부의 클립들과 연관이 없기 때문에 워크플로우 가변성의 프로파일 보다 작성이 용이하다.

그림 16에서 워크플로우 이름을 명시하는 것을 시작으로 해서 워크플로우 가변성 설정 정보들을 위한 테그들이 나열되고 있다. 크게 <init>, <customer-clip>, <sequence>, 그리고 <return-value>가 있다. 우선 <init>에서는 워크플로우가 시작되는 시점에서 요구되는 정보들을 명시하는데, 예제에서는 워크플로우가 시작될 때의 입력된 파라미터 값을 명시하고 있다. <sequence>는 앞에서 정의된 초기화 값들과 클립들을 이용하여 적합한 워크플로우를 구성하는 것이다. 마지막으로 <return-value>는 워크플로우에서 결과 값으로서 워크플로우 호출자에게 반환해야 하는 값을 명시해 준다.

외부화 기법은 언어에 독립적인 커스터마이즈 방법을 제공하며, 워크플로우를 포함하여 다양한 가변성에 적용할 수 있는 기법이다. 논리 가변성의 경우 컴포넌트 내부의 클립을 사용하지 않기 때문에 더욱 쉽게 구현할 수 있다. 또한 데이타에 의한 논리 가변성의 경우 플러그인 기법에서 제시되었던 데이타 클래스를 클립과 같이 제공하여 처리할 수 있다.

외부화 기법에서는 내부에 클립으로 제공되는 메시지 뿐만 아니라 클래스들을 이용할 수 있기 때문에 플러그인 기법보다 더욱 유연하게 워크플로우를 커스터마이즈 할 수 있다. 본 논문에서 제시하고 있는 논리 및 워크플로우 가변성을 완벽하게 지원하며, 소스 생성 기법을 통해서 컴포넌트의 많은 가변적 부분을 효과적으로 커스터마이즈 할 수 있는 기법이다.

5. 평 가

표 2는 관련 연구에서 다루었던 가변성 구현 기법들과 본 논문에서 제시한 기법들을 비교하였다. 표에 명시된 비교 항목들은 개발이 완료된 이후의 컴포넌트를 대상으로 하였으며, 개개의 항목들은 기존에 가변성 기법들을 비교한 *Heineman*의 논문 [16]에서 제시되었던 비교 항목들을 고려하여 제시하였다.

표 2에서 간략한 표현을 위해 Keepence[3]의 기법은 [keepence99]라 명칭하고 COM 컴포넌트의 가변성 구현 기법[5]은 저자의 이름과 발행 연도를 고려하여 [Yoo00]이라 한다. 또한 각 기법들의 약자는 다음과 같다. [Yoo00]에서 SM은 Static Method, MEC는 Method&ExtComponent, CEC는 Class&ExtComponent, 그리고 EC는 ExtComponent 기법을 의미 한다. [Keepence99]에서 SD는 Single Discriminant, MD는 Multiple Discriminant, 그리고 OP는 Option 기법을 나타낸다. Catalysis에서 IT는 Inheritance and the TempleteMethod, 그리고 PF는 Polymorphism and Forwarding을 의미하며, 본 논문의 선택형, 플러그인, 외부화 기법은 각각 SEL, PLG, EXT으로 표기 하였다.

비교 항목에서 확장성은 개발이 완료된 상태에서 컴포넌트가 추가적으로 다양한 가변치를 지원해줄 수 있는지의 여부를 나타낸다. 수행 시 성능은 가변성을 고려하지 않았을 때의 수행성과 비교할 때 가변성 기법들을 적용한 컴포넌트가 가지는 상대적인 성능을 의미한다. 투명성은 가변적 기능을 사용할 때 사용자가 가변적 요소에 대해 고려해야 하는지의 여부이다. 즉, 가변성을 가진 기능성을 사용할 때에도 가변성을 가지지 않은 기능과 마찬가지로 사용할 수 있다면 사용자는 그 기능이

표 2 가변성 구현 기법들의 비교

○:좋음, △:보통, X:미흡, N/C:고려되지않음/불가능함

기법 / 비교 항목	[Yoo00]				[Keepence99]			Catalysis		본 논문의 기법		
	SM	MEC	CEC	EC	SD	MD	OP	IT	PF	SEL	PLG	EXT
확장성	N/C	○	○	○	N/C	N/C	N/C	N/C	○	N/C	○	○
수행 시 성능	○	△	△	△	○	○	○	○	○	○	○	○
투명성	△	○	○	○	○	○	○	○	○	○	○	○
커스터마이즈 용이성	○	△	△	△	N/C	○	○	N/C	△	○	△	△
블랙 박스 지원	○	○	○	○	○	○	○	○	○	○	○	○
바이너리 컴포넌트 지원	○	○	○	○	X	○	○	X	○	○	○	○
논리 불일치	○	○	○	○	○	○	○	○	○	○	○	○
워크플로우 불일치	N/C	N/C	N/C	N/C	N/C	N/C	○	N/C	N/C	○	○	○
상세 지침 제공	○	○	○	○	X	X	X	△	△	○	○	○

가변성을 가진 것인지 아닌지에 대한 고려 없이 동일하게 사용할 수 있을 것이다. 커스터마이즈 용이성 항목은 존재하는 가변성을 커스터마이즈 하기 위해 개발자가 소요해야 하는 노력이 어느 정도인지에 대한 비교항목이다. 블랙 박스 지원 항목은 커스터마이즈 시에 컴포넌트 내부의 정보가 컴포넌트 구매자에게 감추어지는지의 여부이다. 바이너리 컴포넌트 지원 항목은 개발이 완료되어 배포된 컴포넌트를 커스터마이즈 할 수 있도록 지원하는 지의 여부이다. 논리 불일치와 워크플로우 불일치는 논리 가변성과 워크플로우 가변성을 지원하는가에 대한 여부이다.

확장성에 대한 각 기법의 지원 여부를 살펴 보면 SM, SD, MD, OP IT, 그리고 SEL 기법은 내부에 직접 선택할 수 있는 가변치의 경우를 구현해 놓게 되기 때문에 개발 이후에 추가적인 가변치를 제공하기 위해서는 직접 컴포넌트를 수정해야 하기 때문에 확장성을 가지지 못한다. MEC, CEC, EC, PF, PLG, 그리고 EXT 기법은 컴포넌트 내부에 일부 가변치를 두는 경우도 있지만 대부분 외부로부터 가변치를 받아들일 수 있도록 설계를 하고 있어 추가적인 가변치를 제공해줄 수 있다.

수행 시 성능면에서는 대부분 컴포넌트 내부에 추가적인 한번의 전달 통해서 수행되거나 EXT의 경우에는 추가적인 전달 조차도 없이 가변적인 로직을 수행할 수 있기 때문에 비가변성 메소드와 수행 성능면에서 뒤지지 않는다. 다만 Yoo00의 MEC, CEC, EC기법은 외부 컴포넌트로 메소드 수행을 전달하는 경우가 있기 때문에 상대적으로 성능이 낮아질 수 있다.

투명성의 경우 SM기법 사용시 어떠한 로직을 사용할지에 대해 호출 시 결정해야 하고 이에 따라 가변치 선택을 위한 조건들을 모든 사용자가 미리 알아야 하기 때문에 투명성의 조건에 위배가 되며 단순히 선택하기 위한 파라미터를 추가적으로 보내주어야 하기 때문에 보통으로 평가하였다. 다른 모든 기법들은 수행시간에 미리 특정 가변치가 선택이 되어 있기 때문에 가변성 메소드를 사용함에 있어 추가적인 조치가 필요하지 않다.

커스터마이즈 용이성 항목에서 SD와 IT기법은 모두 설계 시에 사용되는 가변성 설계 기법으로 개발이 완료된 컴포넌트의 경우 커스터마이즈가 불가능하기 때문에 N/C로 평가 되었다. SM, MD, OP, SEL기법은 단순히 가변성 선택을 위한 정보를 제공해 주면 되기 때문에 우수하게 평가 되었으며, MEC, CEC, EC, PF, PLG, EXT의 경우 가변치가 되는 메소드나 알고리즘을 위한 설정 파일들을 개발자가 구현해야 하기 때문에 단순 선택보다 많은 비용이 소요되기 때문에 보통으로 평가하였다.

블랙 박스 지원 항목의 경우 모든 기법들이 컴포넌트 내부를 개발자나 사용자가 몰라도 지침에 따라 커스터마이즈 할 수 있기 때문에 우수하다고 평가를 하였다.

바이너리 컴포넌트 지원 항목에서 SD와 IT기법은 설계 및 개발 시에 커스터마이즈를 제공하며 개발이 완료되고 바이너리 상태로 배포가 된 이후의 커스터마이즈를 위해서는 다시 소스 코드를 수정해야 하기 때문에 미흡하다고 평가하였다. 다른 기법들은 바이너리 상태로 배포된 이후에도 커스터마이즈가 이루어 질 수 있기 때문에 우수하다고 평가하였다.

논리 불일치의 경우 모든 기법들에서 단일 메소드 내부 알고리즘의 다양성을 제공하고 있기 때문에 우수하다고 평가하였다.

워크플로우 불일치의 경우 SEL, PLG, EXT를 제외한 대부분의 기법들에서 아예 고려되지 않았으며 지침도 없기 때문에 N/C로 평가하였으며 다만 OP 기법의 경우 설계상 다른 클래스의 메소드를 선택적으로 사용할 수 있도록 하는 기법이기 때문에 보통으로 평가하였다.

표에서 볼 수 있듯이 본 논문에서 제시된 기법들은 확장성, 블랙 박스 지원성, 그리고 성능적인 측면을 포함한 여러 측면에서 우수함을 알 수 있다. 또한 제시된 세 가지 기법들을 실용적으로 적용하기 위한 EJB및 .NET 플랫폼에서의 상세한 지침들을 제공하고 있다.

6. 결 론

본 논문에서 제시하는 기법들은 컴포넌트를 수정 없이 사용하면서도 고객의 목적에 맞도록 커스터마이즈 할 수 있도록 하는 기법에 초점을 두고 있다. 기존에 제시된 커스터마이즈 기법들이 개념적인 수준의 기법을 제시하거나 컴포넌트 개발 시점에서의 커스터마이즈를 지원하는 것과는 다르게, 논문에서 제시된 기법들은 커스터마이즈의 대상이 개발 완료되어 배포된 컴포넌트이다. 개발이 완료된 컴포넌트를 고객에 맞도록 커스터마이즈 하기 위해서는 개발 시에 가변적인 부분을 고려하여야 한다. 이를 위해서 논문에서는 우선 컴포넌트가 가질 수 있는 가변적 유형과 범위를 제시하였다. 가변성 유형 부분에서는 컴포넌트의 기능적 측면에 초점을 두어 논리와 워크플로우의 두 가지로 가변성을 제시하였으며, 이러한 가변적 특성을 컴포넌트 자체에서 지원할 수 있도록 해주는 선택형, 플러그인, 외부화의 3가지 설계 기법들을 제시하였다. 단순히 개념적인 설계가 아니라 구현을 위한 상세한 지침들과 응용할 수 있는 방법도 보여주었다. 평가 부분에서 볼 수 있듯이 본 논문의 기법은 기존에 제시된 여러 기법들과 비교할 때, 컴포넌트를 이상적인 형태인 블랙 박스로 유지 시켜 주면서도 확장성 있고, 우수한 수행성을 제공하고 있다.

또한 제시된 가변성 분류에 따른 세 가지 기법들은 서로 배타적인 것이 아니며, 각각 적합하게 적용될 수 있는 경우가 있고, 가변성에 따라 복합적으로 적용하여 가변성 구현 방법을 제공한다. 가변적인 업무 알고리즘과 워크플로우를 가지는 도메인에서 재사용 가능한 컴포넌트를 구성할 수 있도록 함으로서 개발 생산성을 높이고 비용을 감소시킬 수 있게 될 것이다. 본 논문에서는 가변성 설계 측면에 초점을 두었으며 향후 가변성 분석에 대한 연구가 진척된다면 재사용성 높은 컴포넌트를 개발하는데 효과적으로 적용될 수 있을 것이다.

참 고 문 헌

[1] Heineman, G. T. and Council, W T., *Component-based Software Engineering*, Addison Wesley, 2001.

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

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

[4] Keepence, B. and Mannion, M., "Using patterns to model variability in product families," IEEE Software, Volume: 16 Issue: 4, pp.102-108, July-Aug. 1999.

[5] 유영란, 박동혁, 김수동, "COM 컴포넌트의 다양성 (Variability) 구현 기법", 한국정보과학회 소프트웨어 공학지, Vol. 27, No. 3, pp. 227-240, 2000년 3월.

[6] OMG, "CORBA Components," Version 3.0, Object Mangement Group, June, 2002, http://www.omg.org.

[7] Van Ommering, R., "The Koala Component Model," *Building Reliable Component-Based Software Systems*, Artech House, pp. 223-236, 2002.

[8] Geyer, L., Becker, M., "On the Influence of Variabilities on the Application Engineering Process of a Product Family," SPLC 2, San Diego, CA, USA, August 19-22, 2002.

[9] Becker, M., Geyer, L., Gilbert, A., and Becker K., "Comprehensive Variability Modelling to Facilitate Efficient Variability Treatment," PFE-4 2001, LNCS 2290, pp. 294 - 303, 2002.

[10] Gacek, C. and Anastasopoules, M., "Implementing Product Line Variabilities," ACM SIGSOFT Software Engineering Notes, Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Volume 26 Issue 3, May 2001.

[11] Sharp, D.C., "Exploiting object technology to support product variability," IEEE Proceedings, Oct. 1999.

[12] Bachmann, F. and Bass L., "Managing variability in software architectures," ACM SIGSOFT Software Engineering Notes, Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Volume 26 Issue 3, May, 2001.

[13] 소동섭, 신석규, 김수동, "컴포넌트 가변성 유형 및 Scope에 대한 정형적 모델", 한국정보과학회논문지 소프트웨어 및 응용, Vol. 30, No. 05, pp. 414-429, 2003년 6월.

[14] Takeshi Inoue, "From Class Libraries to Component-Based Development," ICSE Workshop on Component-Based Software Engineering, Apr. 1998.

[15] Sun Microsystems, Enterprise JavaBeans Specification, Version 2.1, http://java.sun.com/products/ejb/docs.html, Sun, Jun. 2002.

[16] Heineman, G. t., "An Evaluation of Component Adaptation Techniques," 2nd Annual Workshop on Component-Based Software Engineering, May 17-18 1999.

정 광 선

2002년 숭실대학교 컴퓨터학부 공학사
2004년 숭실대학교 컴퓨터학과 공학석사
관심분야는 컴포넌트 개발 방법론, 객체지향 분석 설계, 프레임워크 공학

김 수 동

정보과학회논문지 : 소프트웨어 및 응용
제 31 권 제 1 호 참조