Journal of Korea Multimedia Society Vol. 9, No. 8, August 2006(pp. 1076-1085)

일반성 향상을 위한 가변성 설계 기법 및 커스터마이제이션 기법

김철진[†], 조은숙[††]

요 약

다양한 요구 사항들을 완전하게 만족시켜 주기 위한 컴포넌트의 설계는 매우 어려우며 또한 도메인의 특정화된 업무 로직을 완전하게 수용하는 것은 불가능하다. 이러한 요구 사항을 만족시키기 위해 블랙 박스 보다는 화이트 박스 컴포넌트로 제공될 필요가 있다. 본 논문에서는 다양한 도메인의 요구사항을 수용할 수 있는 장치를 제공하기 위해 컴포넌트의 가변성 설계 기법과 이런 설계 기법을 이용하여 컴포넌트를 커스터마이제이션 하기 위한 기법을 제안한다. 컴포넌트의 가변성은 컴포넌트 개발 과정에서 초기 가변성이 설계되며 가변성 적용을 위해 커스터마이제이션 기법을 이용한다. 본 논문에서는 컴포넌트의 기능 변경을 위한 행위 가변성 설계 기법과 커스터마이제이션 기법을 제안한다. 가변성이 적용된 컴포넌트는 이를 기반으로 한 어플리케이션을 개발하는 과정에서 가변성이 재설계될 수 있으며 이러한 과정을 통해 컴포넌트의 가변성이 진화되고 컴포넌트의 일반성이 더욱 향상될 수 있다.

A Variability Design and Customization Technique for Improving Generality

Kim Chul Jin[†], Cho Eun Sook[††]

ABSTRACT

It is difficult to design a component to satisfy several domain requirements and almost impossible to support a specific business logic completely. To satisfy this requirement, there need white box components rather than black box components. So, in this paper, we propose the variability design technique and the customization technique that can support the various requirements of domains. The initial variability of the component is designed at the CD (component development) phase and the customization technique is used for further application of the variability. In the paper, the behavior variability design and customization techniques are used for altering component behavior. As the components that have been developed using the variability technique can easily be re designed during the development of the applications, the variability of the components can be generalized further.

Key words: Behavior Variability(행위 가변성), Customization(커스터마이제이션), Component(컴포넌트), Generality(일반성)

1. 서 론

컴포넌트는 빠른 시간 내에 어플리케이션을 개발하기 위한 개발 블록(Building Block)으로 컴포넌트 이용자(Component User)들에게 컴포넌트 인터페이스(Component Interface)와 컴포넌트 명세(Component

※ 교신저자(Corresponding Author) : 김철진, 주소 : 서울시 강남구 대치4동 890 유니온스틸 빌딩 12층(135-524), 전화 : 02)2191-4677, FAX : 02)2191-4848,
E-mail : chuljin777.kim@samsung.com

접수일 : 2005년 2월 27일, 완료일 : 2005년 5월 17일
[†] 정회원, 삼성전자 DSC(Digital Solution Center)
[††] 정회원, 서일대학 소프트웨어학과
(E-mail : escho@seoil.ac.kr)

Specification) 을 제공한다[1,2,3]. 컴포넌트 인터페이스를 이용해 다양한 도메인의 요구 사항을 충족시키기 위해서는 컴포넌트 내부에 다양성을 제공해야 한다. 그러나 이러한 다양성을 제공하기 위한 가변성을 설계하기가 어려우며 설계된 가변성을 적용하기 위한 기법들이 존재하지 않기 때문에 가변성 부분에 대해 컴포넌트를 공개하여 내부를 변경하는 실정이다[4,5]. 따라서 본 논문에서는 In-House 컴포넌트 개발 시 가변성을 설계하기 위한 기법을 제안한다. 본 기법에 의해 개발된 컴포넌트는 블랙박스 처럼 컴포넌트 내부는 변경하지 않고 컴포넌트 인터페이스 만을 통해 컴포넌트 행위의 변경이 가능하도록 하기 위한 기법이다. 기존의 기법들은 이렇게 인터페이스를 활용하여 컴포넌트의 행위를 변경하는 기법을 제시하고 있지 않다. 또한, 객체 지향 기법을 인터페이스에 이용하고 있지만, 컴포넌트의 행위 변경에 인터페이스를 이용한다는 측면에서 새로운 시도라고 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 CBD 연구들에서 제시하는 가변성 설계 기법의 한계점들을 제시한다. 3장에서는 본 논문에서 제안하는 가변성 설계 기법 및 커스터마이제이션 기법에 대해 설명한다. 4장에서는 제안한 기법의 적용을 통해 얻은 결과에 대한 평가와 기존 기법들과의 차이점을 제시한다. 마지막으로 5장에서 결론 및 향후 연구과제를 제시한다.

2. 관련 연구

Catalysis의 프로세스는 요구사항 분석, 시스템 스펙, 아키텍쳐 설계, 그리고 컴포넌트 내부 설계 단계로 구성되어 있다[6]. 각각의 단계에 정의된 절차 중에 컴포넌트를 이용하여 어플리케이션을 개발할 경우 다양한 도메인에 적용될 수 있도록 가변성을 추출하여 컴포넌트를 개발하기 위한 절차를 정의하고 있지 않다. 부분적으로 "Required Interface"를 정의하는 절차를 포함하고 있는데 이 인터페이스가 가변성을 정의하기 위한 요소가 될 수 있다. 그러나 이 인터페이스를 통한 가변성은 컴포넌트 내부의 변경보다는 외부에서 다양하게 제공될 변경 요소라고 볼 수 있다(그림 1).

Componentware의 프로세스는 분석, 비즈니스 설계, 기술 설계, 스펙, 그리고 구현 단계로 구성되어 있다. Componentware에서는 이러한 단계를 조합하여 다양한 형태의 프로세스를 구성할 수 있도록 프로세스 패턴들을 제공한다[7]. Componentware도 Catalysis의 "Required Interface"와 유사한 "Import Interface"를 제공하고 있다(그림 2). 이 인터페이스도 컴포넌트 외부에서 제공되어 인터페이스로 다양한 요구사항을 충족시킬 수 있도록 한다. 그러나 Componentware에서도 정확한 절차가 제시되어 있지 않으며 컴포넌트 내부 가변성에 대한 설계 기법이나 대응 방안을 고려하고 있지 않다.

커스터마이제이션 기법에 관련된 연구로서 본 논문과 비교되는 논문은 "Using Component Composition for Self-customizable Systems"[8]와 "Component Interface Pattern" [9]가 있다. 그러나 두 논문은 모두 복합 컴포넌트를 개발할 때 커스터마이제이션을 하기 위한 기법들을 다루고 있으며 컴포넌트 내부의

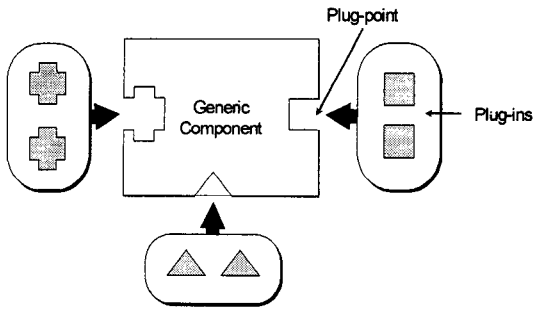


그림 1. Plug-Point & Plug-Ins

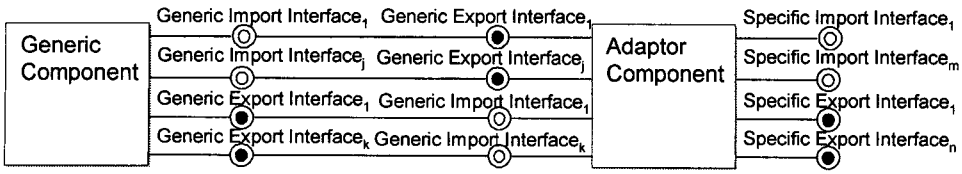


그림 2. Adaptor 컴포넌트

행위에 대한 커스터마이제이션 기법은 미흡하다.[8]의 논문은 컴포넌트들 간에 예상하지 못한 변경을 하기 위해 요구되는 인터페이스에 대한 정의를 하여 동적으로 커스터마이제이션이 되도록 하기 위한 기법이다.[9] 기법도 또한 컴포넌트를 조합하기 위해 컴포넌트의 인터페이스에 제공 인터페이스와 요구 인터페이스 뿐만 아니라 구조적 설명(Structural Description)과 행위적 명세(Behavior Specification)을 통해 표현한다. 구조적 설명은 컴포넌트 인터페이스의 서비스 접근 포인트인 채널(Communication Channel)를 이용하여 표현하며 행위적 명세는 패트리 넷(Petri net)을 이용하여 컴포넌트들 간의 연결과 협업을 명세한다. [8]과 [9]의 연구는 모두 복합 컴포넌트를 동적으로 조합하기 위한 명세 기법 및 패턴을 제시하고 있으며 단일 컴포넌트에 대한 내부 행위 커스터마이제이션에 기법은 미흡하다.

3. 행위 가변성 설계 및 커스터마이제이션 기법

이 장에서는 컴포넌트 내의 행위에 대한 가변성 설계 기법과 커스터마이제이션 기법을 제안한다. 이와 같은 가변성 설계 기법 및 커스터마이제이션 기법은 컴포넌트 개발 단계와 컴포넌트를 이용하여 어플리케이션을 개발하는 단계에 모두 적용될 수 있다.

3.1 행위 가변성 설계 기법

행위 가변성(Behavior Variability)은 컴포넌트 인터페이스 함수의 변경과 컴포넌트 내의 클래스 함수의 변경을 의미한다. 컴포넌트 인터페이스 함수의 변경은 컴포넌트 내의 클래스들을 호출하는 기능에 대한 변경을 의미하며 컴포넌트 내의 클래스 함수의 변경은 컴포넌트의 로직을 변경하는 것으로 클래스 교체를 의미한다.

3.1.1 기능 추가를 통한 행위 가변성 설계 기법

행위 가변성에 대한 인터페이스 설계 기법은 그림 3과 같이 컴포넌트 인터페이스에 대해 확장성을 제공하여 가변성을 설계한다. 기존 컴포넌트 인터페이스는 인터페이스를 구현한 클래스가 컴포넌트 내의 클래스들을 호출하고 있으며 확장된 컴포넌트 인터페이스는 이러한 호출 기능을 변경할 수 있다.

그림 3과 같이 가변성을 위한 확장된 컴포넌트 인터페이스는 기존 컴포넌트의 인터페이스를 상속(Inheritance) 받으며 실제 구현 클래스("Extended Class")는 기존 컴포넌트 인터페이스 내에 있는 기능들을 사용하기 위해 기존 구현 클래스("Class")를 상속 받는다. 이와 같은 설계 기법은 주로 새로운 API(Application Programming Interface)를 추가하거나 기존 함수에 대해 오버로딩(Overloading)해야 하는 경우에 적용되는 설계 기법이다.

3.1.2 기능 변경을 통한 행위 가변성 설계 기법

행위 인터페이스 설계 기법으로 기존 함수의 오버라이딩(Overriding)을 위한 설계는 그림 4과 같이 인터페이스("Interface")는 상속 받지 않고 기존 구현 클래스("Class")만 상속을 받아 설계한다. 확장된 인터페이스 클래스("Extended Class")는 기존 컴포넌트 인터페이스를 사용하기 때문에 따로 정의하지 않으며 컴포넌트 내의 기능을 사용하기 위해 기존 구현 클래스는 상속을 받아야 한다.

동일한 시그니처(Signature)에 다른 기능을 추가하는 이러한 설계 기법은 어플리케이션에서 컴포넌

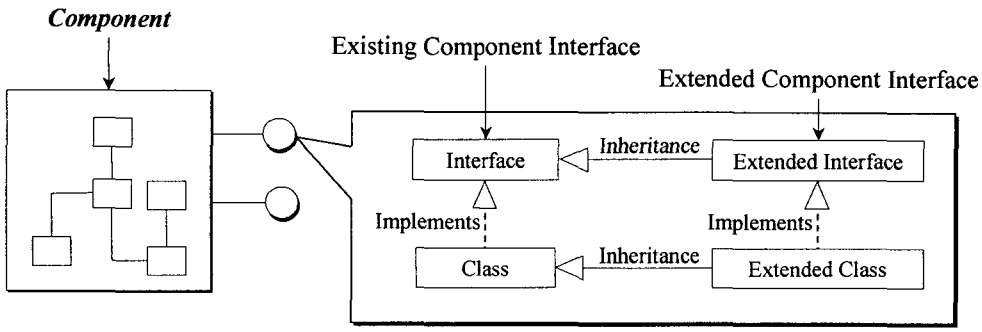


그림 3. 행위 가변성 설계 기법(기능 추가)

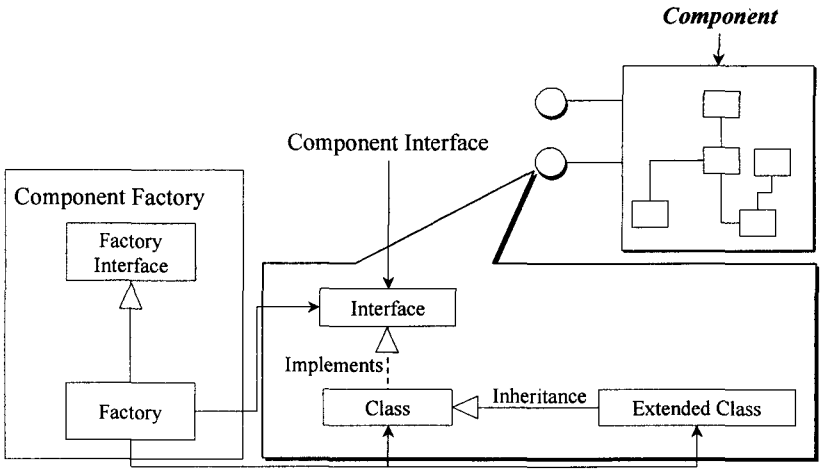


그림 4. 행위 가변성 설계 기법(기존 기능 변경)

트를 사용할 때 원하는 기능을 가진 인터페이스("Class" 또는 "Extended Class")를 선택하도록 설계해야 한다. 즉, 기존 구현 클래스나 확장된 구현 클래스를 사용할 수 있도록 팩토리(factory)를 제공하여 어플리케이션 개발자들이 선택적으로 사용할 수 있도록 한다. 서로 다른 구현 클래스를 통해 컴포넌트 내부의 기능을 호출하더라도 인터페이스는 하나이기 때문에 새로운 기능 추가에 관계없이 다른 부분(컴포넌트 내부 또는 어플리케이션)에는 전혀 영향을 주지 않는다. 그림 4와 같이 팩토리 클래스("Concrete Factory")는 원하는 기능에 따라 "Class"나 "Extended Class" 중에 하나를 선택하여 사용할 수 있다.

3.1.3 클래스 교체를 통한 행위 가변성 설계 기법

행위 가변성 설계 기법 중에 컴포넌트 내의 클래스 기능 변경에 대한 설계 기법을 그림 5와 같이 제시한다. 클래스의 기능을 변경하기 위해서는 다른 클래스로 교체하여 기능을 변경한다. 이렇게 클래스를 교체 가능하도록 하기 위해서는 첫째, 컴포넌트 내부의 클래스에 대해 인터페이스를 정의하여 설계해야 하며 둘째, 해당 클래스를 교체하기 위한 컴포넌트 인터페이스를 정의해야 한다. 이와 같이 클래스를 교체하기 위해서는 교체 클래스의 인터페이스 명세에 맞게 구현하여 입력한다.

앞의 3가지 행위 가변성 설계 기법은 컴포넌트의 내부를 전혀 건드리지 않고 블랙 박스(Black Box) 형태로 변경이 가능하도록 설계하기 위한 기법들이다. 이러한 행위 가변성 설계 기법은 컴포넌트가 어플리케이션에서 이용될 때 컴포넌트 인터페이스나 컴포넌트 내의 클래스가 변경될 수 있도록 하기 위한 기법을 제공한다.

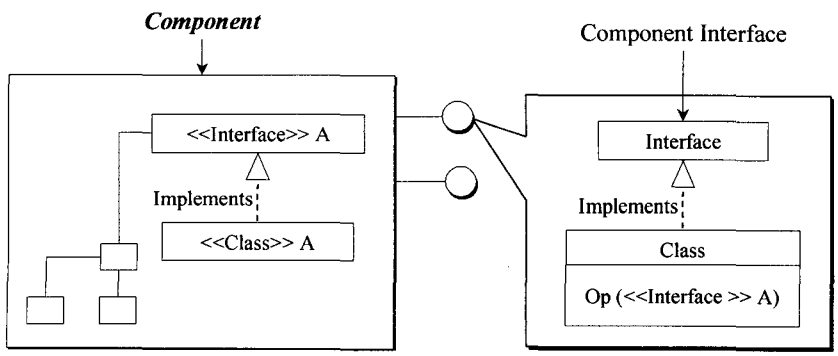


그림 5. 행위 가변성 설계 기법(클래스 교체)

3.2 행위 커스터마이제이션 기법

본장에서는 행위 가변성의 범위에 해당하는 기능 추가, 기능 변경, 그리고 클래스 교체에 대한 설계 기법을 통해 각각의 가변성들을 커스터마이제이션 하기 위한 기법을 제시한다.

3.2.1 기능 추가를 위한 커스터마이제이션 기법

그림 1에서 정의된 설계 기법에 따라 기존 인터페이스를 상속하여 기능을 추가할 수 있다.

코드 1은 컴포넌트 인터페이스로서 기능을 확장하지 않을 경우 "CompIF" 인터페이스를 사용하여 컴포넌트 내의 기능을 사용할 수 있다. 그러나 컴포넌트 개발 시 가변성을 확장 하도록 설계하였거나 어플리케이션 개발 시 이용 컴포넌트의 기능을 확장하여 설계하는 경우, 코드 2와 기존 컴포넌트 인터페이스를 상속하여 기능을 추가한다.

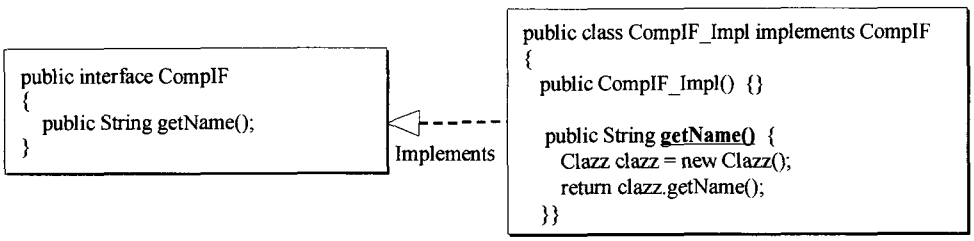
코드 2와 같이 확장된 인터페이스 "CompExtIF"는 새로운 기능인 "getNickName()"을 추가하여 구현한다. 구현 클래스 "CompExtIF_Impl"은 컴포넌트 내의 기능을 사용하기 위해 기존 컴포넌트 인터페이스의 구현 클래스 "CompIF_Impl"을 상속 받았으며 내부적으로 컴포넌트 내의 기능인 "getName()"을 사용하고 있다. 이와 같은 기법을 이용하여 기존 컴포넌트는 전혀 변경을 가하지 않고 확장이 가능하다.

코드 3은 컴포넌트를 이용하여 어플리케이션을 개발한 예를 보여주고 있다. (a)는 기존 컴포넌트 만을 이용하여 어플리케이션을 개발하는 사례이고, (b)는 기능을 추가하여 커스터마이즈된 컴포넌트를 이용하여 어플리케이션을 개발한 사례이다. 확장된 경우의 (b) 사례에서와 같이 컴포넌트에 기능이 추가되면 컴포넌트 인터페이스를 확장된 인터페이스로 사용해야 한다.

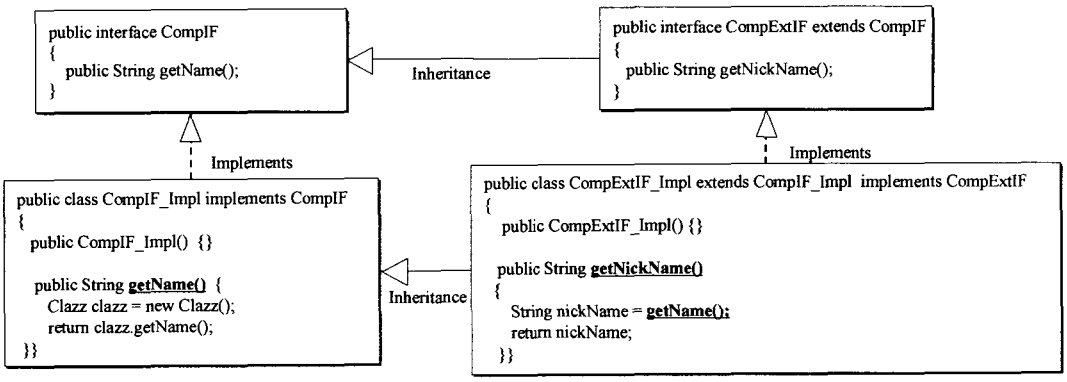
3.2.2 기능 변경을 위한 커스터마이제이션 기법

그림 2에서 정의된 설계 기법에 따라 기존 인터페이스의 기능을 변경할 수 있다.

코드 4는 기존 컴포넌트의 기능을 변경 하기 위해 컴포넌트 인터페이스가 커스터마이즈된 경우를 보여준다. 기존 컴포넌트의 기능인 "getName()"에 대해 변경을 하기 위해 확장된 클래스 "CompExtIF_Impl"은 컴포넌트 구현 클래스 "CompIF_Impl"을 상속받아 "getName()" 기능을 오버라이딩하여 재 구현한



코드 1. 컴포넌트 인터페이스



코드 2. 기능 확장을 위한 컴포넌트 인터페이스

```
public class Application1
{
  public Application1(){ }

  public static void main(String[] args)
  {
     CompIF compIf = new CompIF_Impl();
     String  nickName = compIf.getName();
  }
}
```

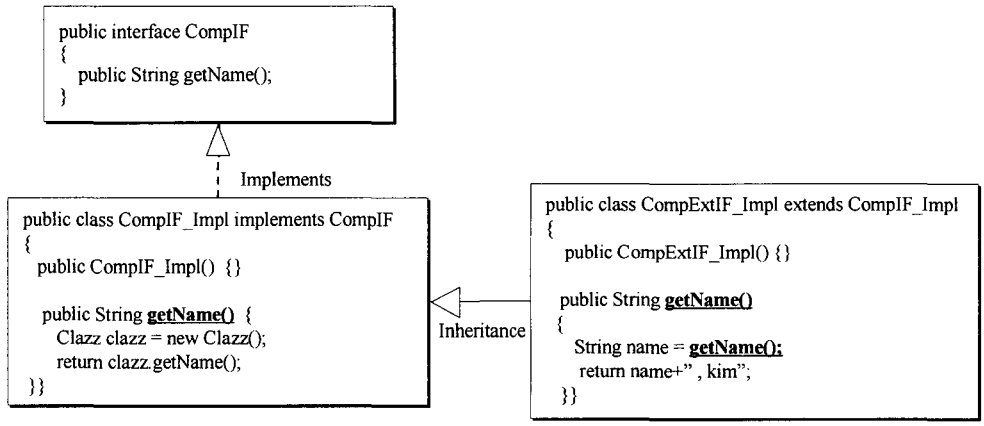
(a) Use the existing component interface

```
public class Application2
{
  public Application2(){ }

  public static void main(String[] args)
  {
     CompExtIF compIf = new CompExtIF_Impl();
     String  nickName = compIf.getNickName();
  }
}
```

(b) Use the extended component interface

코드 3. 기능 확장을 위한 컴포넌트 인터페이스 이용



코드 4. 기능 변경을 위한 컴포넌트 인터페이스

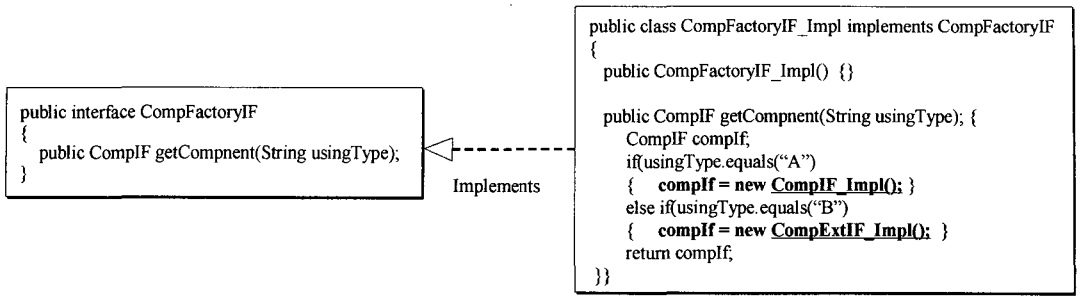
다. 확장된 클래스는 새로운 기능을 추가하지 않기 때문에 인터페이스를 상속 받지 않으며 기존 구현 클래스와 확장된 구현 클래스들은 모두 어플리케이션에서 사용될 때 기존 컴포넌트 인터페이스 "CompIF"로 캐스팅(Casting)하여 사용된다.

팩토리 패턴(Factory Pattern)을 이용해 인터페이스가 확장될 때 마다 팩토리에 컴포넌트 인터페이스를 등록하며 추가된 인터페이스 중에서 어플리케이션에서 원하는 기능의 인터페이스를 선택하여 사용할 수 있도록 한다. 컴포넌트 팩토리는 컴포넌트의 인터페이스를 관리하는 클래스로서 컴포넌트 내부 설계와 분리하여 컴포넌트 외부에서 설계가 되며 클래스를 추가될 때는 코드 수준에서 팩토리에 추가한다.

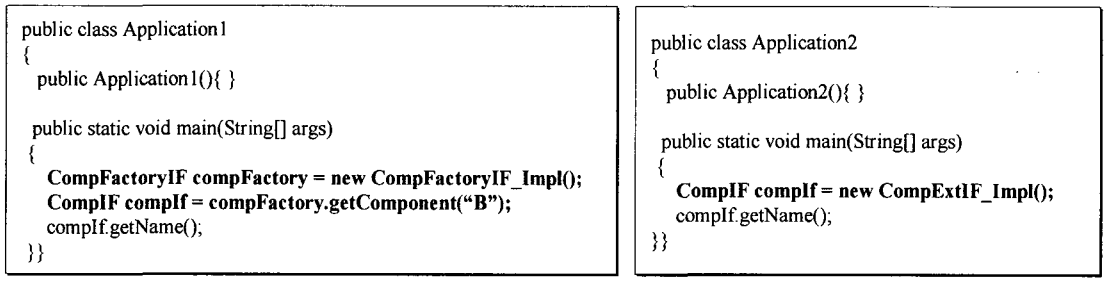
코드 5와 같이 컴포넌트 팩토리 클래스("CompFactoryIF_Impl")에서 "usingType"에 따라 새로운 컴포넌트 인터페이스를 추가한다. 추가될 때 인터페이스의 구현 클래스들("CompIF_Impl", "CompExtIF_Impl")은 서로 다르지만 대리 역할을 하는 인터페이스는 "CompIF"로 동일하다. 이와 같이 하나의 인터페이스("CompIF")로 관리되기 때문에 컴포넌트를 이용하는 어플리케이션은 새롭게 컴포넌트 인터페이스가 추가되더라도 전혀 영향을 받지 않고 개발될 수 있다.

컴포넌트 팩토리는 컴포넌트를 사용하는 도메인의 특성에 따라 다르게 설계 될 수 있으며 팩토리를 설계하지 않고 확장된 인터페이스만을 이용할 수도 있다.

코드 6의 (a)는 컴포넌트 팩토리를 사용하여 어플리케이션을 개발한 경우이며 (b)는 컴포넌트 팩토리를 사용하지 않고 직접 확장된 컴포넌트 인터페이스를 호출하여 개발한 경우이다. 어플리케이션의 특성에 따라 컴포넌트 인터페이스를 다양하게 호출하여 사용할 수도 있으며 어떤 어플리케이션에서는 특정

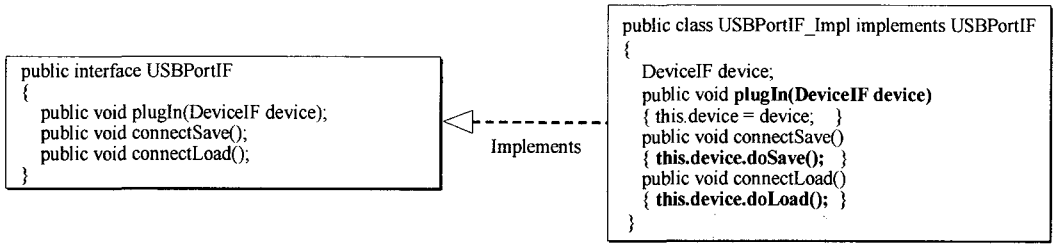


코드 5. 컴포넌트 인터페이스를 위한 컴포넌트 팩토리



(a) Use Component Factory (b) Without Component Factory

코드 6. 기능 변경을 위한 컴포넌트 팩토리 이용



코드 7. 클래스 교체를 위한 컴포넌트 인터페이스

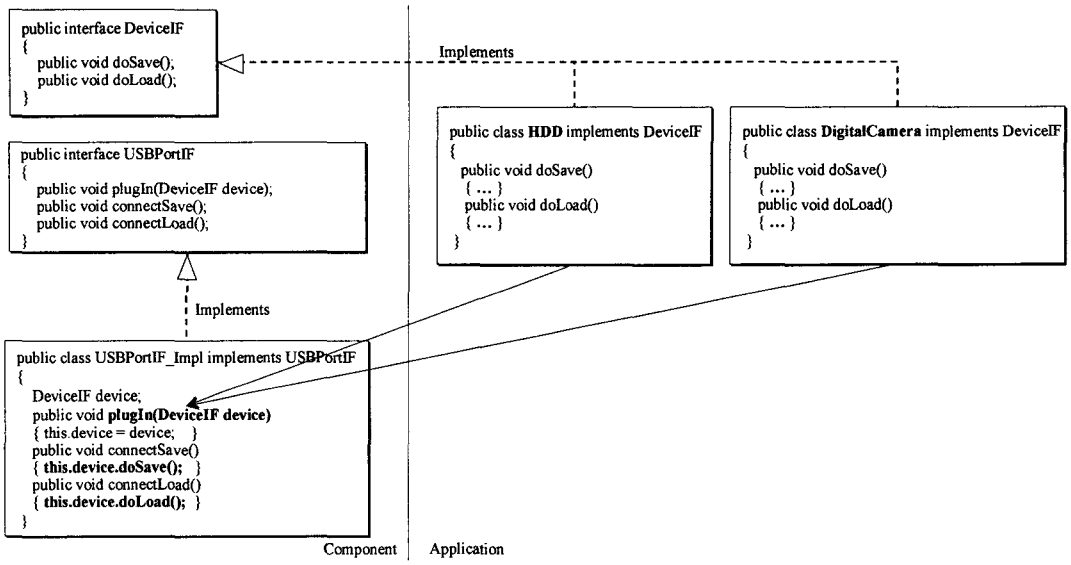
인터페이스만을 고정적으로 사용할 수도 있기 때문에 두 가지 형태 모두 이용될 수 있다.

3.2.3 클래스 교체를 위한 커스터마이제이션 기법

그림 3에서 설명한 것과 같이 가변성이 있는 클래스는 컴포넌트 내에서 교체가 가능하도록 설계되어야 하며 컴포넌트 인터페이스는 교체가 가능하도록 함수를 제공해야 한다.

코드 7의 컴포넌트 인터페이스에서 "plugIn()" 함수는 컴포넌트 내의 클래스를 교체하기 위한 기능이다. 입력 매개변수로 "DeviceIF"라는 인터페이스를 받고 있으며 실제로 이 "DeviceIF" 인터페이스가 구현된다. 앞의 가변성 설계 기법에서 클래스를 교체하기 위한 기본 규칙으로 정의된 대로 교체 클래스는 인터페이스를 정의해야 하며 클래스를 교체하기 위한 컴포넌트 인터페이스를 정의해야 한다. 컴포넌트 내부에서는 "DeviceIF" 인터페이스의 명세에 따라 로직이 구현되어 있으므로 인터페이스의 명세을 따르는 어떠한 구현 클래스라도 교체되는 경우 변경 없이 처리된다.

코드 8에서와 같이 컴포넌트 인터페이스의 클래스 교체 함수인 "plugIn()"으로 클래스를 입력하여



코드 8. 클래스 교체를 위한 컴포넌트 인터페이스 이용

교체할 수 있다. 이러한 교체가 가능한 클래스인 "HDD"와 "DigitalCamera"는 "DeviceIF"의 스펙을 따라 구현해야 한다. 이와 같이 클래스를 교체하기 위해서는 컴포넌트 내부에 교체 가능한 클래스의 인터페이스를 정의해야 하며 외부에서 그 인터페이스의 명세에 따라 구현하여 입력한다.

지금까지 행위 가변성에 대한 설계 기법과 그 설계 기법을 이용한 커스터마이제이션 기법을 제안하였다. 이러한 기법들은 모두 컴포넌트 내부를 변경하지 않고 블랙박스 처럼 컴포넌트를 이용할 수 있는 기반을 제공한다고 할 수 있다.

4. 평 가

이 장에서는 본 논문에서 제안한 가변성 설계 기법 및 커스터마제이션 기법이 다른 컴포넌트 개발 방법론에서 제안한 기법들과 어떤 차이가 있는지 비교 평가한다.

표 1에서와 같이 여러 가지 가변성 설계를 위한 기법들이 제안되어 있지만 본 논문의 기법과는 다르게 상세 설계 기법을 제안하는 것이 아니라 언급만하고 있거나 [8]과 [9]의 연구에서처럼 복합 컴포넌트를 구성할 때의 가변성 설계 기법을 제안하고 있다. 컴포넌트 인터페이스 변경을 위해 Catalysis와 Componentware는 가변성에 대해 언급하고 있지만 구체적인 설계 기법을 제시하고 있지 않으며, [9]의 연구는 Adaptor를 이용해 인터페이스 추가의 기능을 제공하고 있으나, 컴포넌트 내의 행위를 변경하기 위한 기법이 아니라 복합 컴포넌트들 간의 연결을 위한 변경을 의미한다. 그리고 컴포넌트 내의 클래스

표 1. 가변성 특징 및 기능 비교

Factors		This Method	Catalysis	Component-ware	[8] Method	[9] Method
가변성 추출 ?		N	M	M	N	N
가변성 설계 기법 ?		S(단일)	M	M	S(복합)	S(복합)
Component Coupling 축소 ?		S	S	S	S	S
컴포넌트 행위	Interface 변경	S	N	N	N	S(복합)
	Interface 추가	S	N	N	N	S
	Component내의 Class 교체	S	M	N	N	N

[S] Support [N] No Support [M] Mention [단일] 단일 컴포넌트 [복합] 복합 컴포넌트

를 교체하기 위한 기법은 Catalysis에서 'Plug-Ins' 라고 언급하고 있지만, 본 연구에서 처럼 구체적인 Plug-In 기법 을 제시하고 있지 않다. 이와 같이 본 논문에서 제안하는 기법과 같은 컴포넌트 내의 행위를 커스터마이즈하기 위한 연구가 부분적으로만 이루어지고 있음을 알 수 있다.

5. 결 론

지금까지 컴포넌트의 행위에 대한 가변성 설계 기법과 커스터마이제이션 기법을 제안하였다. 본 논문에서 제안한 기법은 기존 컴포넌트를 이용해 소프트웨어를 개발할 때 컴포넌트들 간의 강한 결합도를 최소화할 수 있으며 컴포넌트를 블랙박스로 이용할 수 있도록 해준다. 또한 모든 도메인 요구 사항을 완전하게 충족할 수 있는 컴포넌트를 개발하는 것은 불가능하지만 본 기법을 통해 다양한 요구 사항을 수용할 수 있도록 하여 컴포넌트의 일반성을 더욱 향상시킬 수 있다.

향후 컴포넌트 내의 메시지 흐름 가변성에 대한 설계 기법의 연구가 필요하다. 또한 메시지 흐름에서 특정 컴포넌트의 메시지를 삭제했을 때 관련된 컴포넌트들의 영향을 분석하여 안전하게 변경될 수 있는 설계 기법과 커스터마이제이션 기법의 연구도 필요하다.

참 고 문 헌

[1] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison Wesley Longman, Reading, Mass., 1998.

[2] K. Kang, Issues in Component-Based Software Engineering, *International Workshop on Component-Based Software Engineering*, http://selab.postech.ac.kr/publication/1999_Issues in Component-Based Software Engineering.pdf, 1999.

[3] F. Bachman, *Technical Report*, CMU/SEI-2000-TR-008. ESC-TR-2000-007.

[4] D.F. D'souza and A.C. Wills, *Objects, Component, and frameworks with UML : the Catalysis approach*, Addison Wesley Longman, Inc., 1999.

[5] F. Bachmann, Bass Len, http://www.sei.cmu.edu/plp/variability.pdf, "Managing Variability in Software Architecture," *Software Engineering Institute(SEI)*, 2001.

[6] D.F. D'souza and A.C. Wills, *Objects, Components, and Components with UML*, Addison-Wesley, 1998.

[7] A. Rausch. "Software Evolution in COMPONENTWARE Using Requirements/ Assurances Contracts," *Proceedings of the 22th International Conference on Software Engineering*, pp. 147-156, 2000.

[8] I. Sora, P. Verbaeten, and Y. Berbers, "Using Component Composition for Self-Customizable Systems," *In Proceedings of Workshop On Component-Based Software Engineering: Composing Systems from Components*, pages 23-26. IEEE, 2002.

[9] R. P. e Silva, et al., *Component Interface Pattern*, Procs. Pattern Languages of Program, 1999.

김 철 진

1996년 경기대학교 전자계산학과 이학사

1998년 숭실대학교 대학원 컴퓨터학부 공학석사

2004년 숭실대학교 대학원 컴퓨터학부 공학박사

2004년 카톨릭대학교 컴퓨터 정보공학부 강의전담교수

2004년~현재 삼성전자 DSC(Digital Solution Center) 책임연구원

관심분야 : 컴포넌트 기반 소프트웨어 공학, 컴포넌트 커스터마이제이션, 개발 방법론

조 은 숙

1993년 동의대학교 전자통계학과 이학사
1996년 숭실대학교 대학원 컴퓨터학부 공학석사
2000년 숭실대학교 대학원 컴퓨터학부 공학박사
2002년~2003년 한국전자통신연구원 초빙연구원
2000년~현재 서일대학 소프트웨어학과 전임강사
관심분야 : 개발방법론, 컴포넌트 기반소프트웨어 공학, 소프트웨어 아키텍쳐, 유비쿼터스 컴퓨팅