컴포넌트 개조 지원 도구의 설계 및 구현

김 정 아[†]·권 오 천[††]·최 유 희[†††]·신 규 상[††††]·윤 심[†††††]

요 약

본 연구에서는 컴포넌트의 개조에 필요한 기법과 이를 지원하는 도구를 개발하였다. 컴포넌트를 재사용 하거나 조립하는 과정 중에 컴포넌트의 개조가 필요하게 되는데, 이는 컴포넌트의 인터페이스가 조립하고자 하는 다른 컴포넌트와 다른 경우가 많기 때문이다. 가끔은 새로운 요구 사항에 의해 추가적인 속성의 정의가 필요한 경우도 생기게 된다. 그러므로 컴포넌트의 재사용과 조립에는 컴포넌트 개조의 과정이 필수적이다. 본 연구에서는 컴포넌트 개조를 지원하기 위해서 바이너리 컴포넌트 개조 기법과 개조 컴포넌트에 의한 개조 기법을 제안하였다. 또한 효과적 개조 과정 지원을 위해 개조 지원 도구를 개발하였다. 이로써 소스코드가 없는 기존의 컴포넌트를 개조하여 새로운 요구 사항에 부합되지 못하거나 기존의 다른 컴포넌트와 조립에 문제가 있는 컴포넌트를 쉽게 개조할 수 있도록 지원하였다.

Design and Implementation of Component Adaptation Supporting Tool

Jeong Ah Kim[†] · Oh Cheon Kwon[††] · Hee You Choi[†††]
Gyu Sang Shin[††††] · Shim Yoon[†††††]

ABSTRACT

In this research, the technique and tool for the adaptation of components are suggested. While reusing a component or assembling components, component adaptation should be required since the interfaces of component to be assembled might not be exactly matched. Sometimes, other attributes are needed for new business features or even the same business concept. So, in reusing or assembling a component, component adaptation techniques are essentially required. In this research, we proposed the following : Component Adaptation by Binary Component Adaptation Techniques ; and Component Adaptation by Adaptation Components. Also, we constructed a component adaptation supporting tool. As the results, we can adapt the existing components without source code and can reuse the existing components when the components do not meet new requirements or can not be directly connected with other components to be integrated.

키워드 : 컴포넌트(Component), 개조(Adaptation), 조립(Assembly), 개조 도구(Adaptation Tool)

1. 서 론

컴포넌트 기반 소프트웨어 개발(CBSD : Component Based S/W Development)이 재사용을 통한 소프트웨어 개발의 실질적인 방법으로 인식되면서 소프트웨어 공학 분야의 새로운 기대를 모으고 있다. 이러한 CBSD의 목적은 새로운 어플리케이션을 개발하고자 할 때 검증된 컴포넌트를 재사용함으로써 개발기간을 단축하고 품질을 향상하고자 하는 것이다. 그러므로 CBSD 기반 어플리케이션 개발은, 처음부터 소프트웨어를 개발하기보다는 컴포넌트를 선택(Selection), 개조(Adaptation) 및 조립(Assembly)의 과정을 통해 이루어진다. 컴포넌트를 단순하게 본다면 컴포넌트는 어플리케이션 플러그 인(plug-In) 방식으로 활용되어, '있는 그대로(As-is)방식'으로 재사용(Black-Box Reuse)될 수 있다[2]. 그러나 많은 연구에서 밝혀진 바와 같이 '있는 그대로의 재사용'이란 매우 어렵고 대부분의 어플리케이션 개발 과정에서 실현성이 없는 것으로 보고되고 있다. 즉 어플리케이션의 요구 사항에 부합되는 컴포넌트가 되기 위해서는 컴포넌트는 어떤 방식으로든 개조가 필요한 것이 일반적이다. 특히, 자체적으로 개발한 컴포넌트가 아니고 다른 곳에서 개발된 컴포넌트를 구입 또는 임차하여 재 사용하는 경우 컴포넌트 사용자 입장에서는 그 컴포넌트가 필요한 응용 시스템을 위해 요구되는 기능들을 완벽하게 만족한다고 보장하기 어렵다. 조립 시에 나타나는 가장 빈번한 문제는 구입한 컴포넌트의 인터페이스가 현 어플리케이션 개발에서 요구되는 인터페이스와 다르거나, 인터페이스는 동일하나

† 종신회원 : 관동대학교 컴퓨터 교육과 교수
†† 정 회 원 : 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W·컨텐츠기술연구부 컴포넌트공학연구팀 책임연구원
††† 정 회 원 : 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W·컨텐츠기술연구부 컴포넌트공학연구팀 연구원
†††† 정 회 원 : 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W·컨텐츠기술연구부 컴포넌트공학연구팀 팀장
††††† 정 회 원 : 삼성 SDS, 정보기술연구소 부장
논문접수 : 2002년 6월 11일, 심사완료 : 2002년 8월 6일

실제의 행위가 요구하는 기능과 상이한 경우이다[4]. 이러한 문제점을 극복하는 방법으로는 1) 통합 할 때 기존의 컴포넌트나 시스템을 변경하거나 2) 구입한 컴포넌트를 수정하거나 3) 기존의 시스템과 컴포넌트 사이에 개조기(Adaptor)를 두어 해결하는 것이 일반적이다.

컴포넌트를 개조해야 한다는 관점으로 본다면 기존의 재사용에서의 재사용 단위를 개조하는 것과 같은 맥락으로 볼 수 있을 것이다. 기존의 재사용 단위의 개조는 상속이나 복사를 통한 수정과 같은 white-box 수정과 Wrapping기법과 같은 black-box 수정 방식으로 구분할 수 있다. white-box 수정 기법은 일반적으로 재사용 단위의 내부 구현까지 이해하고 이를 바탕으로, 수정이 필요한 부분을 식별하여 수정하는 노력으로, 객체 지향 개발과 프레임워크 개발에서는 상속에 의해 이루어졌다. 이에 비해 black-box 수정 기법은 재사용 대상의 인터페이스 대한 이해를 바탕으로 수정이 이루어진다[2]. 그러나 CBSD의 특성상 상속에 의해 새로운 클래스를 생성하거나 수정하는 것은 불가능하고 공개된 인터페이스를 통해서만 개조가 가능하다. 특히 객체지향 프로그래밍에서는 클래스의 설계와 수정이 동일한 개발자에 의해 이루어지는 것을 전제로 하고 있지만, CBSD에서는 개발자와 개조자가 다른 것이 일반적이다[11, 12]. 이런 여러 가지 특징에 의해 컴포넌트의 개조는 지금까지의 객체지향 프로그램에서의 클래스 수정과는 다른 기법이 필요하다[10, 11].

본 연구에서는 컴포넌트의 black-box 재사용을 지원하는 컴포넌트 개조 기법을 정의하고 이를 지원하는 도구 개발을 목표로 한다. 제2장에서는 개조를 위한 요구사항 및 기존의 기법들을 정리하고 제3장에서 본 논문이 제안하는 개조 기법을 소개하며 제4장에서 이를 지원하는 시스템의 설계 및 구현 내용을 제시하고 제5장에서 결론을 내린다.

2. 개조 기법의 필요성과 기존의 개조 기법 정의

본 장에서는 컴포넌트 재사용과 관련하여 컴포넌트 개조, 재정의 및 합성에 대해 정의하고, 본 연구의 범위인 개조 기법과 개조 기법이 만족해야 하는 요구 사항을 정의한다.

2.1 개조, 수정, 조립에 대한 정의

컴포넌트 기반 소프트웨어 재사용의 기법은 개조, 수정, 조립으로 구분한다[1, 3]. 필요한 컴포넌트 개조, 수정 및 조립의 개념 및 방법을 정의한다.

- 컴포넌트 수정 : 수정은 컴포넌트의 기본 특성 중의 하나인 프라퍼티(Property)에 의해 이루어지는 컴포넌트의 변경 과정을 의미한다. 프라퍼티란 컴포넌트의 범위와 행위를 결정짓는 특성으로 컴포넌트를 재 사용하여 어플리케이션을 설계 할때 그 값을 변경할 수 있다.
- 컴포넌트 조립 : 컴포넌트를 재 사용하기 위한 가장 자연스러운 과정으로 컴포넌트들끼리 합성(Composition)하는 경우와 다른 어플리케이션들과 통합(Integration)을 의미한다.
- 컴포넌트 개조(Adaptation) : 컴포넌트가 어떤 이유로든 미리 정의된 인터페이스나 행위를 변경해야 할 경우를 의미한다. 간단하게는 인터페이스의 이름이 변경되거나 파라미터의 형식이 변경되는 것을 들 수 있다. 복잡하게는 새로운 인터페이스를 추가하는 경우도 있을 수 있다.

컴포넌트 개조는 일반적으로 다음의 방법으로 가능하다.

① 개조기(Adaptor) 기법 : 인터페이스의 차이를 보이는 두 컴포넌트 사이에 개조기 패턴을 활용하여 개조기 컴포넌트를 정의하여 해결한다. 이 개조기 컴포넌트가 차이나는 두 컴포넌트 사이의 인터페이스 불일치를 해결한다.

② Wrapper 기법 : 이는 인터페이스의 차이를 없애는 새로운 컴포넌트 정의하고 이 컴포넌트가 기존의 컴포넌트를 포함하여 관리하는 방식이다[6,7]. 지금의 바이너리 컴포넌트 기술에 있어서 가장 많이 사용되는 기술로서 Wrapping 컴포넌트는 아주 적은 변경이 필요한 경우에 생성되고, 기존의 컴포넌트에 정의된 기능이 필요하면 요구를 전달하기만 한다. 즉, Wrapping은 이름의 변경 또는 파라미터의 변경 등의 간단한 변경에 있어서 적용 가능하다. 그러나 빈번한 개조가 일어날 경우 개조된 컴포넌트의 크기가 기하 급수적으로 커질 수 있다는 단점 또한 있다.

③ Superimposition에 의한 개조 : 기본 개념은 개조를 해야 하는 컴포넌트와 개조를 처리하는 컴포넌트를 두 개로 분리하되, 둘을 하나로 묶어서 긴밀하게 동작시키도록 하려는 것이다[5]. 이는 개조의 대상이 되는 컴포넌트와 개조를 처리하는 컴포넌트 모두 독립적으로 재 사용하려는 목적을 갖는다. 이 방법은 컴포넌트의 개조 유형이 다양하지 않고 정해진 범위 내에서 이루어진 다는 전제 하에 가능한 방법이다. 이를 위해 언어를 처리하는 새로운 모델을 제시하였으며, 이 기법의 지원을 위해서는 기존의 언어에서 이루어지는 메소드 호출을 별도로 처리하는 기반 시스템을 구축해야 하기 때문에 일반적이지 못하다.

④ Binary Adaptation 기법 : UCSB(University of California at Santa Barbara)에서 개발한 방법으로 동적 로더(Dynamic Loader)를 개발하여, 로딩된 바이너리 컴포넌트를 직접 수정하는 방식이다[8, 9]. 이를 위하여

Java의 클래스 로더를 수정하여 UC Santa Barbara 대학에서 개발한 Delta file 컴파일러를 사용해야 한다는 제약점이 있다. 즉, 일반적인 환경에서는 사용 불가능하다. 이러한 개념이 자바에서는 가능하나, EJB(Enterprise JavaBeans) 컴포넌트의 경우는 로더를 변경하는 것이 명세에 금지되어 있으므로 EJB에 바로 적용하기는 어려운 기법이다.

⑤ Adaptation Component에의한 기법 : 컴포넌트의 개조 유형별로 처리에 필요한 컴포넌트를 제공하고, 기존의 컴포넌트와 개조 컴포넌트를 감싸는(Wrapping) 기법으로 본 연구에서 제안하는 방법이다.

⑥ 컴포넌트 자체를 수정하는 기법 : 바이너리 컴포넌트로부터 컴포넌트의 reflection 기법을 이용하여 컴포넌트 구현 내용을 역으로 추출 후 컴포넌트 코드 자체를 수정하여 새로운 컴포넌트를 만들어 내는 방법으로 본 연구에서 제안하는 방법이다.

2.2 컴포넌트 개조의 필요성

컴포넌트를 재 사용하는 과정에서 필연적으로 부딪히는 두 가지 문제가 개조와 인터페이스의 진화(Evolution)이다[9]. 컴포넌트를 사용하는 예를 들어 문제를 먼저 파악해 보자. 컴포넌트를 사용하는 두 개의 어플리케이션 A, B를 가정한다. 각각은 서로 다른 회사로부터 컴포넌트를 구입하였다. 문제를 간단하게 살펴보기 위하여 어플리케이션의 클래스 인터페이스를 다음과 같이 정의한다.

```
class A {
    public void output(PrintStream os) ;
}
class B {
    public void print( ) ;
}
```

두 클래스 모두는 프린팅을 제공하고 있지만 메소드의 시그네이처(Signature)가 서로 다르다. 그러나 이런 이름이 다른 것조차도 두 클래스를 하나의 어플리케이션으로 통합하기 어렵게 한다. 예를 들어서 A와 B로부터 상속받은 컴포넌트를 저장한 후 그 안에 저장된 내용을 일괄적으로 출력하고자 한다면 아래의 코드와 같아진다.

```
Enumeration e ;
while (e.hasMoreElements( )) {
    Printable p = (Printable)e.nextElement( ) ;
    p.print( ) ;
}
```

그러나 이러한 코드는 서로간의 시그네이처가 다르므로 허용되지 않는다. 만약 우리가 클래스 코드를 수정할 수 있다면 별 문제가 아니겠지만, 바이너리 컴포넌트에 있어서는 코드의 수정이 불가능하기 때문에 인터페이스의 사소한 차이 역시 통합에 막대한 영향을 주게 된다.

또 다른 문제는 인터페이스가 지속적으로 진화하는데서 오는 문제이다. 진화란 개발 과정에서 컴포넌트의 인터페이스의 첨삭이 필요하게 되기 때문에 발생한다. 새로운 요구사항으로 인해 새로운 기능이 추가되거나 발견된 오류를 수정하는 과정에서 변화가 일어날 수 있다. 인터페이스의 추가는 새로운 구현의 추가를 수반하게 되고, 구현부가 추가되면 전체 컴포넌트의 재 컴파일을 필요로 한다. 이러한 변경은 이미 컴파일된 코드에는 불필요한 작업이므로, 인터페이스의 진화 역시 추가적인 부담이 필요한 작업이다.

마지막 개조의 필요성은 컴포넌트의 의미적 명세가 달라지는 경우이다. 예를 들어서 Java 컴포넌트중의 하나인 Date의 경우는 시간과 날짜를 표현하는 기능을 제공하며, 이는 시스템 종속적 의미 해석방법을 지니고 있다. 날짜는 1에서 31의 범위를 갖고 달은 0에서 11까지로 정의되며 0이 1월을 의미한다. 그러나 이러한 날짜 처리 방법이 적합하지 않은 어플리케이션도 존재할 수 있을 것이다. 이때 컴포넌트 재이용자가 1을 2월로 5를 6월로 해석하여 프로그램을 할 수도 있겠지만, 근본적으로 월을 구분하는 기준을 변경하려면 컴포넌트의 구현이 변경되어야 한다. 이때 개조의 이유는 구현 메소드의 변경이다.

2.4 컴포넌트 개조 기법의 요구 사항

컴포넌트 개조기법이 만족해야하는 사항을 정의하면 다음과 같다[11].

- Transparent : 컴포넌트가 Transparent하다는 것은 개조 전의 원 컴포넌트(Original Component)를 사용하는 개발자나 개조후의 컴포넌트를 사용하는 개발자 모두에게 개조의 사실이 알려져서는 안된다는 것이다. 뿐만 아니라 개조가 필요 없는 컴포넌트는 당연히 추가적인 노력 없이 접근 가능해야 한다.
- Black-Box : 소프트웨어 개발자는 컴포넌트를 재사용하기 전에 컴포넌트에 대한 기능성을 충분히 파악해야 하지만, 가능하면 최소한의 정보만으로 가능해야 한다. 이러한 요구 사항은 컴포넌트의 개조과정에 원 컴포넌트가 어떻게 구현되었는지에 대한 정보가 활용되지 않음을 가정한 것이다. 즉, 인터페이스를 통한 컴포넌트의 개조만으로 국한한다.
- Composable : 개조된 컴포넌트 역시 다른 컴포넌트와 다시 합성될 수 있어야 하며 개조된 컴포넌트는 다시 개조될 수도 있어야 한다.
- Homogeneous : 기존의 컴포넌트를 사용하는 코드는 여

전히 개조된 컴포넌트를 사용할 수 있어야 한다.

- Ignorant : 기존의 컴포넌트는 개조에 대한 아무런 사전 지식이 없어야 한다.
- Identity : 기존의 컴포넌트는 개조와 관계없이 독립적으로 식별 가능해야 한다. 즉, 추후의 또 다른 개조에 사용 될 수 있어야 한다.
- Architectural focus : 기존의 컴포넌트와 개조된 컴포넌트를 포함하여 개발할 어플리케이션의 아키텍처에 대한 명세를 제공할 수 있어야 한다. 이때 기존의 컴포넌트와 개조된 컴포넌트의 명세는 달라야 한다.

3. 컴포넌트 개조 기법

본 연구에서는 컴포넌트 개조를 위한 기법을 크게 두 가지로 구분하여 제안하였다. 첫 번째는 Wrapping의 기법을 보완하기 위한 개조 패턴 컴포넌트에 의한 개조 기법이며, 두 번째는 바이너리 컴포넌트 개조 기법을 EJB 컴포넌트에 적용한 바이너리 코드를 직접 개조하는 기법이다.

본 연구에서는 컴포넌트의 개조가 필요한 일반적인 요인을 다음과 같이 식별하였다.

① 관리항목의 변경(속성의 추가/재명명) : 컴포넌트가 처리해야 하는 속성을 추가하거나 기존의 속성의 이름을 변경하는 것이다.

② 인터페이스에 정의된 오퍼레이션의 시그네이처 변경 : 가장 빈번한 경우로 오퍼레이션의 이름이 변경되거나 오퍼레이션에 정의된 파라미터나 반환 값의 변경이 필요한 경우이다.

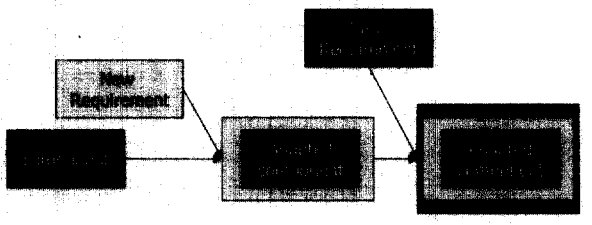
③ 인터페이스에 정의된 오퍼레이션의 추가 : 인터페이스에 새로운 오퍼레이션을 추가하여 서비스의 범위를 확장하는 경우를 의미한다. 인터페이스에 추가된 오퍼레이션에 대해서는 당연히 내부 구현 메소드도 추가되어야 한다.

④ 메소드의 변경 : 이미 정의된 오퍼레이션에 대한 새로운 행위를 정의해야 하는 경우이다.

3.1 개조 컴포넌트에 의한 컴포넌트의 개조

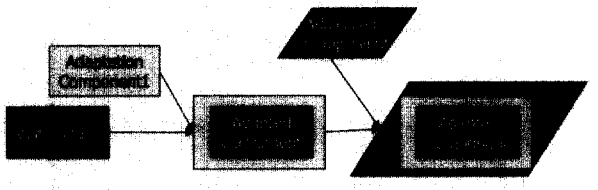
컴포넌트 개조의 기존 방법은 일반적으로 Wrapping에 의한 방법으로 이미 개발된 컴포넌트에 새로운 기능을 추가하여 새롭게 컴포넌트를 구성하는 것이다. 이런 방법을 사용할 경우 (그림 1)에서 보는 바와 같이 컴포넌트의 개조가 필요할 경우마다 기존의 컴포넌트를 새로운 컴포넌트로 Wrapping하게 되면 새로운 필요가 등장할 때마다 컴포넌트가 계속 확장되어지는 문제가 발생한다[5].

본 연구에서는 컴포넌트 개조의 원인을 분석하고 원인별로 필요한 개조 컴포넌트(Adaptation Component)를 제공하여 개조 원인을 해결하는데 필요한 개조 컴포넌트를 한번 Wrapping하면 동일한 원인에 대해서는 더 이상의 wrapping 없이 지속적인 개조를 지원할 수 있도록 하였다. 이는 개조 과정이 반복됨에 따라 컴포넌트가 계속 Wrapping되어지는 필요성을 제거하기 때문에, 컴포넌트 개조 과정의 효율을 증진시킨다. 물론 상이한 개조 유형의 경우는 해당 원인을 처리하는 개조 컴포넌트를 다시 wrapping 해야 한다.



(그림 1) 기존의 Wrapping 방법

개조 컴포넌트를 개발하기 위해서 설계 패턴 개념을 적용하였다. 즉 개조 유형별로 이를 처리하는 패턴을 정의하고 구현하여 개조를 처리하도록 하였다. 개조 컴포넌트는 필요한 메타 모델을 활용하여 새롭게 정의된 요소를 처리한다. (그림 2)는 이런 과정을 보여 준다. 컴포넌트 개조 유형별로 본 연구에서 개발한 개조 컴포넌트와 기존의 재사용 대상이 되는 컴포넌트를 합성하여 새로운 컴포넌트를 제공하는 방법이다. 다른 유형의 개조일때만 Wrapping은 발생한다. 개조된 컴포넌트를 사용하는 사람은 실제로 개조 컴포넌트가 포함되어 있는지를 알지 못하고도 개조된 컴포넌트가 제공하는 인터페이스를 통해서 서비스를 받게 된다.



(그림 2) 개조 컴포넌트(Adaptation Component)에 의한 개조

본 연구에서는 속성에 대한 개조 부분과 인터페이스에 대한 개조 부분을 처리하는 개조 컴포넌트를 개발하였다. 먼저 속성 개조 컴포넌트를 정의하면 다음과 같다.

속성의 추가/삭제를 지원하는 부분으로 이를 처리하는 개조 컴포넌트를 Attributable Component라고 설계하였다. 새로운 이름의 속성을 추가하고자 하는 요구가 발생했을 때 적용하는 패턴이다. 이 개조 컴포넌트가 적용될 수 있는 전제 조건은 다음과 같다.

① 속성의 추가로 기존에 정의된 비즈니스 업무가 달라지지는 않는다.

② 속성의 추가 후에는 Get/Set의 인터페이스 추가를 통

해 컴포넌트의 활용이 확장된다.

③ 속성의 유형 변경은 지원하지 않는다. 이미 정의된 속성의 타입을 변경할 수는 없는데 이는 객체 지향의 상속을 통해서도 속성의 재정의는 정의하지 않는 것과 동일하다.

속성 개조의 기법은 다음과 같이 정의할 수 있다.

① 이미 정의된 속성에 대한 처리 인터페이스는 기존의 컴포넌트를 통해서 제공된다.

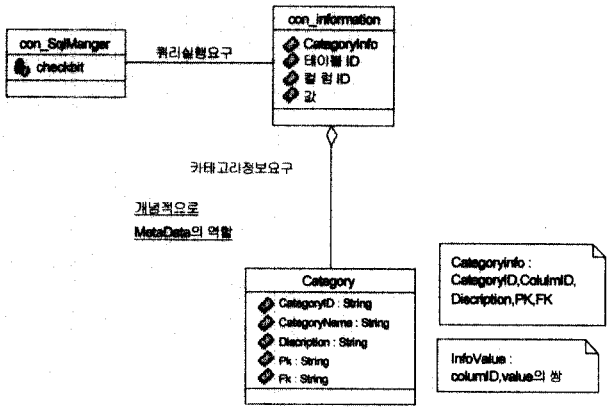
② Attributable 컴포넌트의 추가 후 Attributable 컴포넌트에 의해 새로운 속성의 추가가 이루어진다.

③ Attributable 컴포넌트는 새롭게 정의한 속성을 메타 모델로 정의하고, 이 메타 모델로부터 정보를 저장 및 관리하게 되는데, 영구적 저장소를 요하는 부분의 경우는 이에 대한 데이터 베이스를 생성 관리할 수 있다.

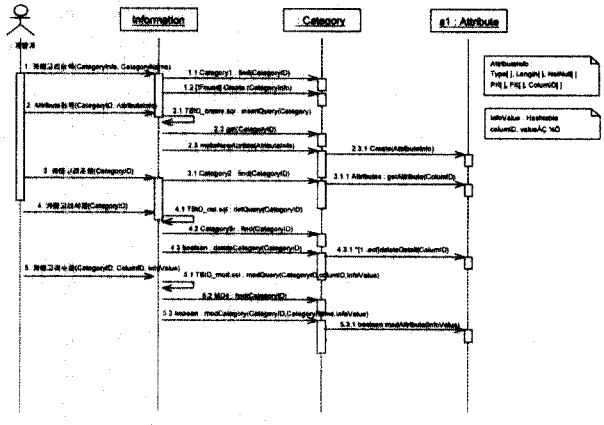
④ 개조된 컴포넌트는 기존의 컴포넌트와 Attributable 컴포넌트를 함께 Wrapping하여 두 컴포넌트의 인터페이스를 다 제공한다.

Attributable Component 자체에 대한 설계는 (그림 3)과 같다. Attributable 컴포넌트의 내부 구조를 보면 컴포넌트에는 추가적으로 정의될 속성에 대한 별도의 정보를 정의할 수 있다. 일반적으로 추가되는 속성의 경우는 Category를 정의할 수 있다. 이는 몇 가지 종류의 속성이 모여서 하나의 의미를 이룰때 정의하는 것으로 필수적인 것은 아니다. 또한 속성은 해당 속성이 영속적 성격을 가진다면, 속성이 저장될 테이블에 대한 정보를 함께 정의하여, 저장 및 검색의 오퍼레이션이 실행 될 수 있도록 한다. 이 컴포넌트는 새로 추가되는 속성에 대한 메타 정보를 관리하는 책임을 갖는다. 각 속성은(name, value)의 쌍으로 정의된다. (그림 3-b)에는 컴포넌트를 구현하는 내부 클래스들간의 상호작용을 설명한 것이다. 사용자는 Attributable을 조작할 수 있도록 제공하는 Information 인터페이스를 통해서 먼저 필요하다면 Category를 등록하고 이 Category안에 정의될 속성을 등록하여 추가된 속성에 대한 메타 모델이 등록되는 과정을 예시하고 있다.

또 다른 지원 유형으로 인터페이스의 개조는 재명명 개조 컴포넌트에 의해 이루어지는데, 재명명이란 컴포넌트에 정의된 인터페이스나 속성의 이름을 변경하는 것으로 본 연구에서는 개조기 패턴[12]를 컴포넌트로 구현하여 처리하도록 하였다. 개조기 패턴을 적용한 재명명 개조용 컴포넌트는 재명명된 정보를 담고 있는 메타 모델을 관리하여, 클라이언트가 개조된 컴포넌트에 정의된 오퍼레이션의 이름으로 호출하면, 재명명 컴포넌트는 메타 모델을 확인하여 이것이 변경된 오퍼레이션인지를 확인하고 변경된 것이면 메타 모델에 정의된 기존의 오퍼레이션 이름을 찾아 그 오퍼레이션 호출 문을 수행하게 하고, 그렇지 않으면 기존의 오퍼레이션 호출을 그대로 사용하도록 구현하였다. 이 개조 컴포넌트 역시 메타 모델을 바탕으로 개조를 처리하도록 설계하였다.



(a) 컴포넌트 내부 구조



(b) Attribute 생성에 대한 OID

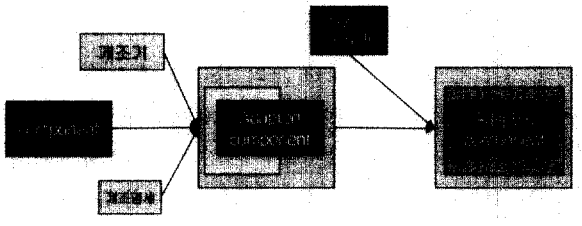
(그림 3) Attribute Component 설계

현재 개조 컴포넌트에 의해 처리 가능한 개조 유형은 관리 항목의 변경 및 인터페이스의 시그네이처 변경 두 가지이다. 즉, 개조용 컴포넌트는 속성 개조 컴포넌트와 재명명 개조 컴포넌트만이 구현되었다. 그 이유는 인터페이스의 추가나 메소드의 추가의 경우는 행위를 재 정의해야 하는데, 이는 개발자에 의해 그때마다 결정되는 사항으로 미리 패턴화 해둘 수 없는 부분이기 때문이다. 이 유형에 패턴 컴포넌트를 적용하려면 컴포넌트 설계시 메소드 자체를 별도의 컴포넌트화 해두어, 변화가 일어날 수 있는 메소드가 기존의 컴포넌트 코드에서 분리되도록 미리 설계되어 있어있는 경우에만 가능하다. 즉 가변성을 지니는 메소드의 경우, 클래스 안에서 직접 구현되기보다는 별도의 Strategy 클래스나 Strategy컴포넌트로 구분시켜 설계해 두어야 한다. 만

약 이렇게 Strategy패턴이 적용된 컴포넌트 설계였다면, 개조 과정에서 Strategy 패턴 컴포넌트의 메타 모델에 이 정보를 저장하고, 개발자가 새로운 Strategy 클래스나 Strategy컴포넌트를 구현하여 Wrapping할 수 있다. 이렇게 개조하면, 개조된 컴포넌트가 동작할 때 행위 개조 컴포넌트가 새롭게 정의된 Strategy 클래스나 Strategy 컴포넌트를 구동시킬 수 있다. 인터페이스의 추가 역시 메소드의 추가를 동반하기 때문에, 메소드 처리와 동일한 이유로 개조 컴포넌트를 모든 경우에 적용할 수 없다는 제한점을 갖는다.

3.2 Binary Component Adaptation 기법

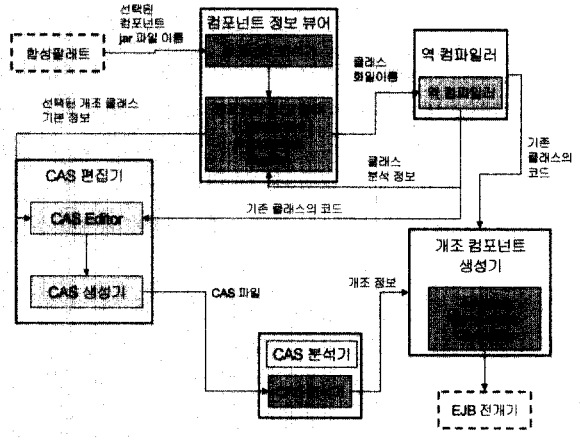
제안한 개조 컴포넌트에 의한 개조는 wrapping에 의해 컴포넌트의 크기가 커지는 문제를 해결할 수 있으나, 속성과 인터페이스의 재명명에 관한 유형을 제외하고는 개조의 모든 유형을 해결하지 못하는 단점이 있었다. 또한 바이너리 컴포넌트의 개조는 소스 코드 없는 상황에서 이루어져야 하는 경우가 많다. 이런 점을 고려하여 본 연구에서는 BCA 기법을 응용한 기법을 바이너리 컴포넌트 개조에 적용해 보았다. 소스 코드를 직접 접근할 수 없는 경우 코드 없이도 개조가 가능하도록 하려는 것이다. EJB에서는 순수 BCA 방법처럼 EJB에 정의된 자바 클래스 로더를 변경할 수는 없으므로 바이너리 코드를 개조한 새로운 컴포넌트로 결국은 wrapping을 하는 것이 되지만, 기존의 방식과 달리 컴포넌트 개조된 내용 자체를 wrapping하려는 것이 아니라 개조 명세를 기존의 컴포넌트에 추가하고 개조 명세 처리기를 함께 wrapping 하므로써 반복적인 개조가 일어나더라도 개조 명세만이 추가될 뿐 개조된 컴포넌트가 계속 wrapping에 의해 크기가 증가하는 문제는 해결할 수 있다.



(그림 4) 개조 명세를 기반으로 한 개조 기법

(그림 4)의 예를 보면 기존의 컴포넌트를 개조하기 위하여 개조 명세를 작성하고 이를 처리하는 개조기와 함께 Wrapping을 한다. 추후 새로운 개조가 필요시, 기존의 개조 명세를 수정하여 새로운 개조 컴포넌트를 만들 수 있다. 또한 처음 개조가 이루어질 때 한번 Wrapping된 개조기는 계속 재사용될 수 있고, 개조된 컴포넌트에 대한 개조의 경우는 개조 명세의 개조의 개념이 된다.

제안한 바이너리 컴포넌트 개조를 위해서는 개조에 필요한 정보를 정의하는 명세를 작성해야 한다. 컴포넌트 재사용자가 컴포넌트에 추가되거나 변경되어야 하는 내용을 정의된 명세 구문 형식에 따라 기술하면 컴포넌트 작성 도구가 인식하는 컴포넌트 명세 형태로 변형된 후 컴포넌트 개조 도구에 의해 일반 컴포넌트로 만들어진다. 기존의 BCA에서는 인터페이스나 메소드의 추가와 재명명의 경우만을 처리하도록 지원하지만, 본 연구에서는 새로운 속성의 추가 부분까지 처리 할 수 있도록 확장하였다. 본 연구에서 개발한 컴포넌트 개조기는 기존의 컴포넌트에 새로운 애트리뷰트, 인터페이스, 메소드를 추가하거나 이름 변경 등과 같은 원인에 의한 개조를 모두 지원한다. 이를 위하여 기존 컴포넌트를 역 컴파일(Decompile) 하여 소스 코드를 얻고 이 소스 코드와 관련하여 작성된 컴포넌트 개조 명세를 합쳐 새롭게 개조된 컴포넌트를 생성한다. 이를 바탕으로 한 컴포넌트 개조기의 설계안은 (그림 5)와 같다.



(그림 5) 컴포넌트 개조기 시스템 설계

- 컴포넌트 정보 뷰어 : 패키지에 포함된 클래스와 인터페이스의 목록을 확인하고 각각의 클래스와 인터페이스에 대한 정보를 제공한다.
- 역 컴파일러 : 컴포넌트 정보 뷰어에 선택된 개조 대상의 바이너리 코드로부터 클래스 이해에 필요한 각종 정보와 소스 코드를 얻어낸다.
- CAS 편집기 : CAS(Component Adaptation Specification) 편집기는 컴포넌트 개조가 필요할 때, 개조의 원인별로 필요한 정보를 입력받아, 컴포넌트를 구성하는 클래스 또는 인터페이스별 CAS 파일을 생성한다.
- 개조 컴포넌트 생성기 : CAS 파일의 내용을 기존의 컴포넌트에 반영하여 새로운 코드를 만들고 이를 패키지화한다. CAS 생성기로 만들어진 CAS 명세 파일을 CAS 분석기에 의해 분석하여 실제 개조된 정보를 식별하여 이를 역컴파일러가 만들어낸 기존의 코드에 반

영하여 새로운 코드를 생성하고, 이를 전개기에 전달하여 EJB 패키지를 만들어 낸다

컴포넌트 개조의 전반적인 시스템 운영 흐름을 살펴 보면 다음과 같다. 개조를 위해 가장 먼저 이루어져야 하는 부분은 개조가 필요한 대상 컴포넌트를 이해 하는 과정이다. 이를 위해 이미 작성된 컴포넌트들 중 개조가 필요한 컴포넌트를 선택하고, 역컴파일러에 의해 컴포넌트에 대한 소스 코드를 얻고, Reflection과 Introspection 기법을 이용하여 컴포넌트에 등록된 클래스나 인터페이스 정보를 분석하여 컴포넌트 정보 뷰어에게 제공한다. 컴포넌트 정보 뷰어는 분석된 정보를 바탕으로 컴포넌트 안에 정의된 클래스 및 인터페이스 목록을 제공하고 또한 각 클래스나 인터페이스에 정의된 정적 정보를 제공하여 개조자의 컴포넌트 이해 과정을 지원한다. 이러한 이해의 과정을 통해 개조가 필요한 컴포넌트에 정의된 클래스나 인터페이스에 갖는 애트리뷰트나 오퍼레이션을 확인하고 인터페이스 불일치 또는 알고리즘의 대체 및 인터페이스의 추가가 필요한지를 판단하여, 각 경우에 해당하는 정보를 CAS 편집기를 통해 입력하여 CAS 파일을 생성한다. CAS는 개조에 필요한 명세로 CAS는 개조 유형별로 별도의 명세 구문을 갖고 있는데, 개조 유형은 인터페이스에 오퍼레이션을 추가, 추가된 오퍼레이션에 대한 메소드 정의, 속성 이름의 재명명, 인터페이스의 재명명, 기존에 정의된 구현의 변경이다. 이들 각각에 대한 개조 명세 구문을 통해 개조의 범위를 파악할 수 있다. 개조 명세는 태그를 갖는 언어 형태로 정의하였다.

- JAR 정보
  - <JARNAME> </END> : 태그 사이에 Jar File의 Full Path와 Jar File Name을 입력하여 cas 파일 처리시에 개조 대상 컴포넌트로의 참조자로 관리한다.

- 인터페이스에 오퍼레이션 추가 명세
  - <ADD ABSTRACT METHOD></END> : 인터페이스가 추가될 클래스의 패케지와 이름이 입력된다.
  - <NEW INTERFACE NAME></END> : 추가될 인터페이스의 이름이 입력된다.
  - <ACCESS MODIFIER></END> : 추가될 인터페이스의 접근 제한자가 입력된다.
  - <PARAMETER LIST></END> : 추가될 인터페이스의 파라미터가 입력된다.
  - <EXCEPTION THROW></END> : 추가될 인터페이스의 예외 발생이 입력된다.
  - <RETURN TYPE></END> : 추가될 인터페이스의 반환 타입이 입력된다.
  - <LINE></END> : 새로 생성된 소스에 추가된 인터페이스의 라인수가 입력된다. 컴파일 에러발생시 해당 위치로 이동할때 참조한다.

- 속성 추가 명세
  - <ADD ATTRIBUTE></END> : 애트리뷰트가 추가될 클래스의 패케지와 이름이 입력된다.
  - <NEW ATTRIBUTE NAME></END> : 추가될 애트리뷰트의 이름이 입력된다.
  - <DATA TYPE></END> : 추가될 애트리뷰트의 데이터 타입이 입력된다.
  - <ACCESS MODIFIER></END> : 추가될 애트리뷰트의 접근 제한자가 입력된다.
  - <LINE></END> : 새로 생성된 소스에 추가된 인터페이스의 라인수가 입력된다. 컴파일 에러발생시 해당 위치로 이동할때 참조한다.

- 메소드 추가 명세
  - <ADD METHOD></END> : 메소드가 추가될 클래스의 패케지와 이름이 입력된다.
  - <NEW METHOD NAME></END> : 추가될 메소드의 이름이 입력된다.
  - <NEW METHOD CODE></END> : 추가될 메소드의 코드가 입력된다.
  - <LINE></END> : 새로 생성된 소스에 추가된 인터페이스의 라인수가 입력된다. 컴파일 에발생시 해당 위치로 이동할 때 참조한다. 메소드는 1개 라인 이상이기 때문에 시작 라인과 마지막 라인으로 입력된다.

- 메소드 대체 명세
  - <REPLACE METHOD></END> : 변경될 메소드 클래스의 패케지와 이름이 입력된다.
  - <EXIST METHOD NAME></END> : 변경될 메소드의 시그네이처가 입력된다.
  - <NEW METHOD CODE></END> : 변경될 메소드의 코드가 입력된다.
  - <LINE></END> : 새로 생성된 소스에 추가된 인터페이스의 라인수가 입력된다. 컴파일 에러발생시 해당 위치로 이동할 때 참조한다. 메소드가 1개 라인 이상이기 때문에 시작 라인과 마지막 라인으로 입력된다.

- 속성 이름의 재명명 명세
  - <RENAME ATTRIBUTE></END> : 애트리뷰트가 재명명될 클래스의 패케지와 이름이 출력된다.
  - <OLD ATTRIBUTE NAME></END> : 변경될 애트리뷰트의 이름이 입력된다.
  - <NEW ATTRIBUTE NAME></END> : 변경될 애

트리뷰트의 데이터 타입이 입력된다.

- <LINE></END> : 새로 생성된 소스에 변경된 라인 수가 입력된다. 컴파일 에러발생시 해당 위치로 이동할때 참조한다. Rename Attribute는 해당 클래스 외에 개조가능한 모든 클래스를 포함한다.

- 인터페이스 재명명 명세
  - <RENAME INTERFACE></END> : 인터페이스가 재명명될 클래스의 패케지와 이름이 입력된다.
  - <OLD INTERFACE NAME></END> : 변경 전 인터페이스의 시그네이쳐가 입력된다.
  - <NEW INTERFACE NAME></END> : 변경 후 인터페이스의 시그네이쳐가 입력된다.
  - <LINE></END> : 새로 생성된 소스에 추가된 인터페이스의 라인수가 입력된다. 컴파일 에러발생시 해당 위치로 이동할 때 참조한다.

이런 구문에 의해 작성된 CAS로부터 개조 정보를 분석하여 원래 컴포넌트를 구성하는 개조가 필요한 클래스나 인터페이스를 개조하여 새로운 EJB 패키지로 묶일 클래스와 인터페이스를 생성한다. 컴파일 과정에서 작성된 CAS 파일을 분석하여 개조 정보를 추출하여 이를 개조 컴포넌트 생성기에 전달하면, 개조 컴포넌트 생성기는 역 컴파일러에 의해 생성된 코드와 개조 정보를 통합하여 새로운 클래스 코드를 생성하고 개조된 클래스에 대해서만 컴파일 과정을 수행한다. 컴파일 오류가 없는 경우 외부 시스템은 EJB 전개기(EJB Deployer)를 통하여 새로운 개조 컴포넌트를 생성한다

4. 컴포넌트 개조기 구현

본 연구에서 컴포넌트 개조를 위해 프로토타입으로 설계한 부분은 다음과 같다.

① Binary Component Adaptation(BCA)에 의한 컴포넌트 개조
② 패턴 컴포넌트 의한 컴포넌트 개조

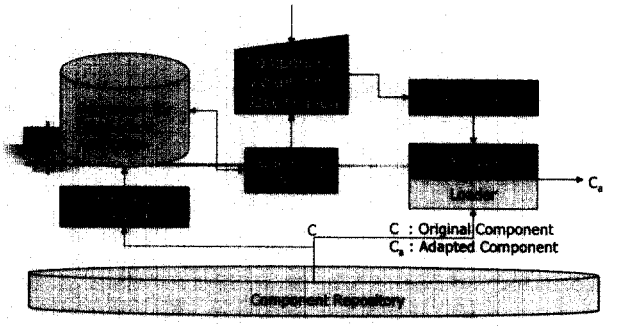
4.1 BCA에 의한 컴포넌트 개조 시스템 개요

컴포넌트 실행 파일을 분석하여 컴포넌트에 대한 기본 정보를 추출하고, 컴포넌트 사용자에 의해 정의된 CAS를 분석하여 컴포넌트 실행 파일 자체를 수정하는 기법을 지원하는 시스템으로 기본적인 흐름은 (그림 6)과 같이 정의할 수 있다.

컴포넌트 저장소(Component Repository)는 공용 컴포넌트 저장소로 재사용 가능한 컴포넌트를 관리, 저장하는 기능을 수행한다. 정보 분석기는 저장소에 저장된 컴포넌트의 실행 파일로부터 컴포넌트 개조에 필요한 정보를 추출한다. 추출되는 정보를 정리하면 아래와 같다.

① 컴포넌트에 정의된 인터페이스 정보
② 컴포넌트에 정의된 클래스 정보
③ 클래스에 정의된 오퍼레이션 정보
④ 클래스에 정의된 속성 정보
⑤ 오퍼레이션에 정의된 메소드 정보

이들 정보를 바탕으로 개조자는 Component Viewer를 통해 추출된 정보를 이해하고 필요한 개조 내용을 CAS로 정의한다. 이는 CAS 컴파일러에 의해 분석되어 컴포넌트의 실행 파일을 직접 수정하게 된다.



(그림 6) BCA에 의한 컴포넌트 개조 시스템 개요

4.2 구현 기능 및 실행 예

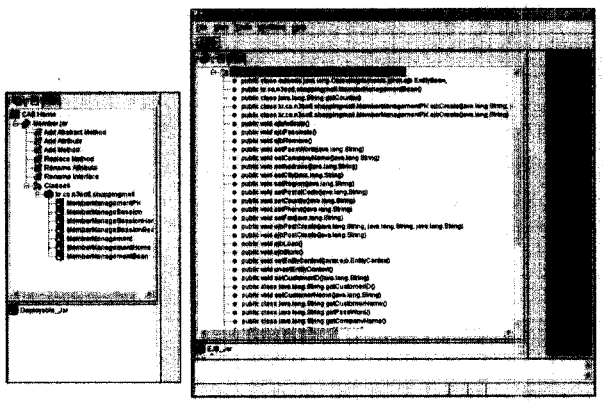
컴포넌트 개조를 위해 구현된 기능을 살펴보면 다음과 같이 정의할 수 있다.

- 컴포넌트 실행 파일 분석기 : 컴포넌트를 구성하는 .class 파일을 분석하여 컴포넌트에 정의된 모델 정보를 추출한다. 컴포넌트를 구성하는 인터페이스 및 컴포넌트를 구성하는 클래스들에 정의된 속성과 메소드에 대한 정보를 분석한다. 이를 바탕으로 실제 .java 클래스를 생성한다.
- 컴포넌트 정보 뷰어 : 역 컴파일된 java 파일을 에디터 상에 보여준다. 컴포넌트에 대한 정적인 정보로 클래스에 정의된 속성 정보와 메소드 정보를 제공한다.
- 컴포넌트 개조명세 작성기 : 컴포넌트 개조에 필요한 내용을 정의한 개조파일을 분석하여 개조의 원인을 파악하고 개조내용을 추출한다. 개조의 내용으로는 속성 추가, 인터페이스 추가, 메소드 재정의, 재명명이 가능하다.
- 컴포넌트 개조명세 컴파일 : 개조에 필요한 내용이 작성된 명세를 컴파일하여 개조명세 상의 문법적 오류를 발견한다.
- 개조 컴포넌트 생성기 : 기존의 컴포넌트에 개조명세

에 작성된 내용을 반영하여 새로운 컴포넌트를 위한 코드를 만들고, 이를 컴파일하여 패키지에 필요한 클래스 파일을 생성한다.

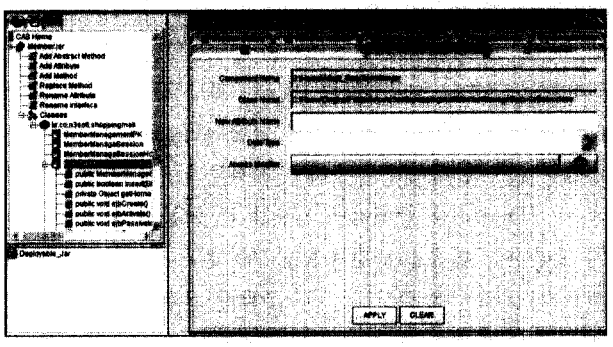
- 개조 컴포넌트 전개기 : 컴포넌트의 특성에 따라서 개조에 필요한 정보를 정의하여 이를 전개기로 넘겨서 컴포넌트 패키지가 생성되도록 한다.

대상 컴포넌트를 선택하면 내부적으로 jar를 풀어 압축된 클래스 파일들을 얻어내게 되는데, 그 결과 컴포넌트를 구성하는 클래스 리스트를 제공한다. 각각의 클래스 파일을

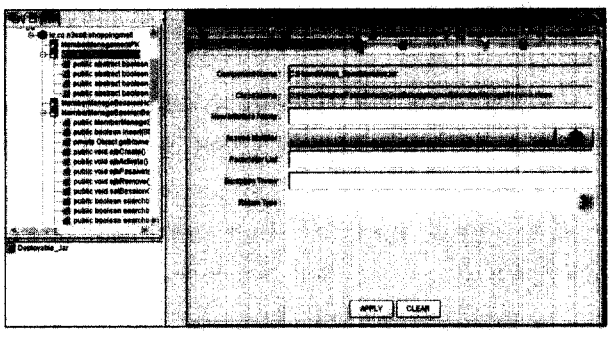


(a) CAS 및 클래스 정보 (b) 엔터티 빈 분석 결과 화면

(그림 7) 컴포넌트 실행 파일 분석기에 의해 얻어진 컴포넌트 정보

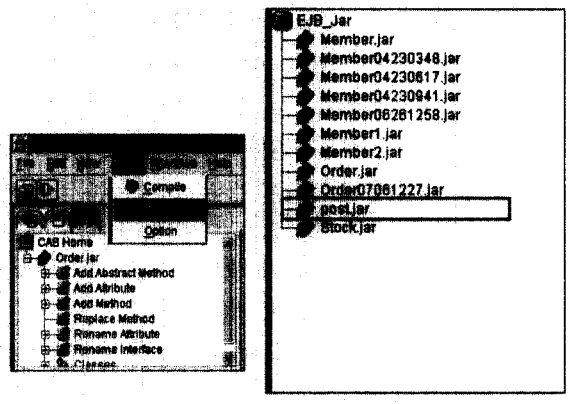


(a) 빈 클래스 선택시 화면



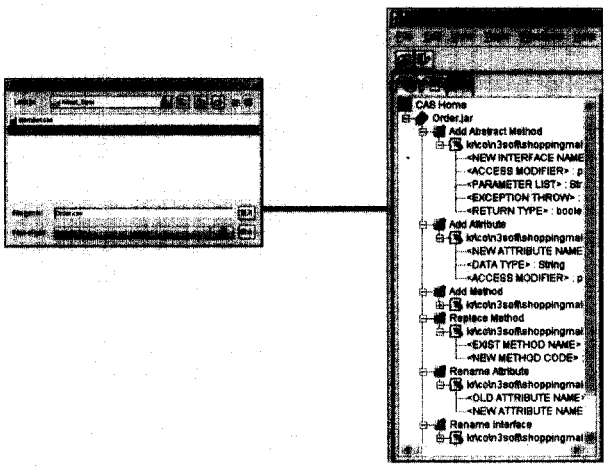
(b) 인터페이스 선택시 화면

(그림 8) CAS 편집기 화면



(a) MareJar 선택화면 (b) 생성된 개조 컴포넌트

(그림 9) 개조 컴포넌트 생성 과정



(a) 개조 컴포넌트의 CAS 명세 선택 (b) 개조 명세 목록

(그림 10)

사용자가 선택하면 역 컴파일러를 이용하여 소스 코드를얻게 된다. 이렇게 얻어진 소스 코드를 분석하여 클래스를 구성하고 있는 멤버 변수와 멤버 메소드에 대한 정보를 얻어내게 된다. 그 결과는 (그림 7)과 같다. 이렇게 제공한 클래스 정보를 바탕으로 컴포넌트에 정의된 인터페이스, 속성정보 등을 이해하고 나서 개조가 필요한 컴포넌트를 선택 한다. 이 선택의 과정은 클래스 목록에서 클래스 이름을 더블클릭하면 되고, 그 결과 (그림 8)과 같은 개조 명세 편집기가 제공된다. 선택한 개조 대상이 클래스인지, 인터페이스인지에 따라 개조의 범위가 파악되고, 해당 범위에 대해서만 개조 할 수 있도록 편집기가 제공된다. 즉, 빈 클래스의 경우는 인터페이스와 관련된 개조는 불가능하며, 인터페이스 클래스일 경우는 메소드나 속성과 관련된 개조는 불가능하기 때문이다. 각각의 개조 유형별로 CAS 편집기가 제공하는 화면에 따라 개조 정보를 입력하면 CAS가 작성되고 CAS 분석기에 의해 개조 명세의 오류의 유무를 확인한 후, 기존의 컴포넌트와 통합하여 새로운 컴포넌트를 생성한다.

이 과정은 (그림 9)에 정의된 것처럼 컴포넌트 전개기에 의해 새로운 컴포넌트로 패캐징 되며, 컴포넌트의 이름은 기본적으로 개조되기 전의 컴포넌트 이름에 개조시각으로 식별자를 두었다. 또한 이미 개조된 컴포넌트라 하더라도 (그림 10)의 CAS 명세의 개조를 통해서 또 다른 개조가 가능하다. 이때는 기존의 개조 컴포넌트에 정의된 CAS를 (그림 10)(a)처럼 선택하면 (그림 10)(b)에서 이미 작성된 명세가 보이며, 이를 편집하여 새로운 CAS를 작성하여 또 다시 다른 개조 컴포넌트를 만들 수도 있도록 구현하였다. 즉, 원래 개조 대상 또한 별도로 개조가 가능하고, 이미 개조를 거친 컴포넌트 역시 진화 과정에서의 개조가 가능해진다.

4.3 평 가

앞에서 컴포넌트 개조가 만족해야 하는 조건을 7가지로 정의하였다. 본 연구가 제안한 방법이 이들을 어떻게 만족하는지에 대한 평가 결과는 다음과 같다.

- Transparent : 새로 만들어진 컴포넌트는 기존의 컴포넌트와 다른 식별자를 갖고, 또한 기존의 컴포넌트를 wrapping 한후 새로운 인터페이스를 제공하므로 개조의 사실을 모르는채 사용할 수 있다.
- Black-Box : 기존의 컴포넌트에 대한 Reflection 기법을 통해 최소한의 이해만으로 컴포넌트를 개조할 수 있도록 지원한다. 이는 컴포넌트 정보 뷰어를 통해 지원하며, 또한 CAS 편집기가 선택한 클래스를 분석하여 가능한 개조 유형을 알려주기 때문에 클래스에 대한 자세한 분석 없이 개조 명세를 작성할 수 있도록 하였다.
- Composable : 개조된 컴포넌트는 다시 동일한 개조 과정을 통해 개조가 가능하며, 다른 컴포넌트와 조립 가능하다.
- Homogeneous : 기존의 컴포넌트와 개조된 컴포넌트는 물리적으로 구분되어 있으므로 기존의 컴포넌트가 제공하는 인터페이스를 통해 사용해온 프로그램은 변화 없이 사용 가능하다.
- Ignorant : 기존의 컴포넌트는 소스 코드가 공개되지 않는 바이너리이므로, 추후 어떻게 개조될 것인지에 대한 아무런 고려 없이 개발되어도 무방하다. 단, 본 연구에서 해결하지 못한 행위의 개조의 경우는 컴포넌트를 설계 할때 추후 개조를 대비하여 유연한 설계로 작성되어 있어야 하나, 이것이 추후 어떤 개조가 발생할지를 미리 안다는 것을 의미하지는 않는다.
- Identity : 기존의 컴포넌트와는 다른 컴포넌트 식별자를 부여하기 때문에, 물리적으로 다르고, 인터페이스의 차이로 인해 논리적으로도 식별 가능한 컴포넌트이다.
- Architectural focus : 새롭게 개조된 컴포넌트는 기존의 컴포넌트를 분석해 만든 명세와 개조 명세를 통합하여 새로운 명세를 제공하고 있다.

이와 같이 본 연구에서 제안한 BCA에 의한 개조와 개조 패턴에 의한 개조는 개발한 지원 도구에 의해 컴포넌트 개조가 만족해야 하는 요구 사항을 만족하고 있으며, 기존에 가장 많이 사용되고 있는 단순한 Wrapping 기법이 갖고 있는 컴포넌트 크기의 지속적 확대의 문제를 해결한다.

5. 결 론

본 연구에서는 바이너리 컴포넌트의 개조 과정을 지원하는 컴포넌트 개조 시스템을 설계하고 개발하는 것을 목표로 하였다. 이를 위하여 먼저 기존의 컴포넌트 개조 기법에 대한 요구 사항을 정의하고, 가능한 기술적 근거를 조사/분석하고, 이를 바탕으로 컴포넌트의 개조 기술을 정립하고 이를 구현하였다.

본 연구에서 접근한 개조 기술은 크게 2가지이다.

- BCA 기법에 의한 컴포넌트 실행 파일 개조
- 개조 컴포넌트에 의한 컴포넌트 개조

본 연구에서는 컴포넌트 개조의 유형을 크게 4가지로 구분하였다. 개조 컴포넌트에 의한 개조에서는 각각 개조 유형별로 개조를 처리하는 컴포넌트를 개발하여, 유형별 개조가 필요할때 기존의 컴포넌트와 개조 컴포넌트를 함께 Wrapping하는 기법이다. 이는 기존의 단순 wrapping의 컴포넌트 크기의 지속적 확대라는 문제를 해결하기는 하지만, 재명명과 속성의 추가에 대해서만 지원할 뿐 모든 개조를 처리하지 못한다는 단점을 발견하였다. 즉, 개조 컴포넌트에 의한 개조의 경우는 컴포넌트의 재구조화가 선행되어야 하는 경우가 있었다. 특히 행위 개조 컴포넌트의 경우는 기존의 컴포넌트가 행위의 변경을 가정한 인터페이스를 정의하지 않았을 경우는 기존의 컴포넌트에 인터페이스를 추가하는 과정을 더 거쳐야 한다.

이를 보완하는 또 다른 방법이 BCA 기법를 응용한 방법으로 실행 파일 자체를 변경하여 CAS에 정의된 개조 내용을 반영하여 실행하는 방법이다. 이는 소스 코드가 제공되지 않는 바이너리 컴포넌트의 개조를 위해 기존의 BCA를 변형한 것으로, 소스 코드 없이 개조 명세를 작성하여 이를 기존의 컴포넌트와 통합하여 새로운 컴포넌트를 생성하는 방법을 구현한 것이다. 이는 순수한 BCA에의한 컴포넌트 개조의 경우는 Java VM(Virtual Machine)을 직접 수정해

야 하는 경우가 발생하기 때문에 EJB에 적용 가능성은 불투명하였고, 순수한 BCA에 의한 컴포넌트 개조의 경우 플랫폼 종속적인 개조 시스템을 개발해야 하는 단점을 해결하였다. 기존의 바이너리 컴포넌트를 분석하여 컴포넌트 구조와 이를 구성하는 각 클래스에 대한 정적인 정보를 얻어내고, 이를 이해 한후, 개조가 필요한 부분을 식별하여, 개조 명세를 작성하면, 개발한 개조기에 의해 기존의 바이너리 컴포넌트를 수정하고, 새로 만들어진 명세를 통합한 개조 컴포넌트 코드를 만들게 된다.

본 연구를 통해 컴포넌트의 재구조화 없이 단순한 인터페이스 변경이 필요한 경우는 개조 컴포넌트에 의한 개조와 BCA 기법의 연구 부산물인 클래스 파일 분석기를 통해 실제 컴포넌트에 대한 코드를 추출하고, 여기에 CAS에 정의된 개조의 내용을 반영하여 새로운 컴포넌트를 생성할 수 있게 되었다. 이로써 다른 회사에서 개발한 컴포넌트나 이미 만들어진 컴포넌트를 효과적으로 개조하여 어플리케이션으로 통합하는 과정을 지원할 수 있게 되었다.

추후 보완 사항으로는 개조 개념 자체를 Refactoring과 연결 시켜 좀더 정형화된 형태로 정의할 필요가 있다. 즉, 개조 개념과 원인별 처리과정을 정형화하여 개조 과정에서 발생할 수 있는 오류를 미리 방지하고, 개조의 자동화 비율을 증대할 수 있을 것으로 기대 한다.

참 고 문 헌

[1] Johannes Sametinger, "Classification of Composition and Interoperation," OOPSLA'96 Poster Presentation.

[2] Bradford Kain J. "Component : The Basics : Enabling an Application or System to be the Sum of its parts," Object Magazine, Vol.6, No.2, pp.64-69, April, 1996.

[3] Nierstrasz Oscar, Meijler Theo Dirk, "Research Directions in Software Composition," ACM Computing Surveys, Vol. 27, No.2, pp.262-264, June, 1995.

[4] Nierstrasz Oscar, Meijler Theo Dirk, " Component-Oriented SOftware Technology," Object-Oriented Software Composition, Prentice-Hall International, pp.3-28, December, 1996.

[5] Jan Bosch. Superimposition : A Component Adaptation Technique. Information and Software Technology, 41(5), pp. 257-273, March, 1999.

[6] Jim Q. Ning, "Component-Based Software Engineering," IEEE Software, 1997.

[7] Jim Q. Ning, "A Component-Based Software Development Model," in Proceedings of 21th Annual International Computer Software and Application Conference, 1996.

[8] Ralph Keller, Urs Holzle, "Implementing Binary Component Adaptation for Java," www.cs.ucsb.edu/oocsb.

[9] Urs Holzle. Integrating Independently-Developed Components in Object-Oriented Languages. Proceedings of ECOOP '93, Springer Verlag LNCS 512, 1993.

[10] George T. Heineman, "A Model for DesigningAdaptable Software Components," in 22th Annual International Computer Software Annual International Conference, Vienna, Austria, August, 1998.

[11] George T. Heineman, "An Evaluation of Component Adaptation Techniques," Computer Science Department, Worcester Polytechnic Institute, WPI-CS-TR-99-04.

[12] Erich Gammar, et al, Design Patterns : Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

김 정 아

e-mail : clara@kwandong.ac.kr

1990년 중앙대학교 전자계산학과 공학 석사

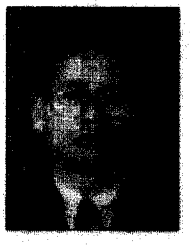
1994년 중앙대학교 컴퓨터공학과 공학 박사

1996년 중앙대학교 객원연구원

1999년 관동대학교 컴퓨터 교육과 조교수

현재 관동대학교 컴퓨터 교육과 부교수

관심분야 : 재사용, CBD, Product Line, MDA 기술, 프로젝트 관리 및 프로세스 개선 등



권 오 천

e-mail : ockwon@etri.re.kr

1994년 영국 Teesside 대학교 대학원 S/W 공학과 공학석사

1998년 영국 Durham 대학교 대학원 전산과학과 공학박사

1991년 미국 RTP IBM 연구소 객원 연구원

1993년 시스템공학연구소 선임연구원

현재 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 S/W · 컨텐츠기술연구부 컴포넌트공학연구팀 책임연구원

관심분야 : 재사용, CBD, Product Line, Web Services, MDA 기술 등

최 유 희

e-mail : yhchoi@etri.re.kr

1999년 부산대학교 컴퓨터공학과(학사)

2001년 부산대학교 컴퓨터공학과(석사)

2001년~현재 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 S/W · 컨텐츠기술연구부 컴포넌트공학연구팀 연구원

관심분야 : 컴포넌트 기반 소프트웨어 개발 방법론, 소프트웨어 아키텍쳐, 분산 컴포넌트, 소프트웨어 재사용 등

신 규 상

e-mail : gsshin@etri.re.kr

1981년 성균관대학교 통계학과 학사

1983년 서울대학교 대학원 계산통계학과 이학석사

2001년 충남대학교 대학원 컴퓨터과학과 이학박사

1983년 시스템공학연구소 연구원

1987년 시스템공학연구소 선임연구원

1997년 한국전자통신연구원 책임연구원

현재 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W·컨텐츠기술연구부 컴포넌트공학연구팀 팀장

관심분야 : CASE, S/W 컴포넌트 기술, 웹 서비스 기술, MDA 기술, 멀티미디어 시스템

윤 심

e-mail : yoonshim@samsung.com

1986년 중앙대학교 전자계산학과 이학사

1992년 프랑스 파리 6 대학 전산과 공학석사

1996년 프랑스 파리 6 대학 전산과 공학박사

1985년~1990년 LG 소프트웨어

1996년~현재 삼성 SDS, 정보기술연구소 부장

관심분야 : 웹 서비스, 지식 관리 등