2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

컴포넌트 합성을 위한 재정의에 관한 연구

염희균*, 김혜미**, 황선명*
*대전대학교 컴퓨터공학과
**전주공업대학 컴퓨터정보과
e-mail:yhg1124@zeus.taejon.ac.kr

A Study on Component Customization for Component Composition

Hee-Gyun Yeom*, Hye-Mee Kim**, Sun-Myung Hwang*
*Dept of Computer Engineering, Daejeon University
**Dept of Computer Engineering, Jeonju Technical College

요약

최근 컴포넌트를 기반으로 하는 소프트웨어 재사용은 합성에 의한 개발 방법론을 근간으로 한다. 여러 컴포넌트를 함께 동작하도록 조립하는 과정이 곧 소프트웨어를 개발하는 과정이 되는 것이다. 본 연구에서는 기존의 코드 기반 또는 문서 기반의 컴포넌트보다는 독립적으로 실행 가능한 블랙박스 개념의 컴포넌트를 기반으로 한 재사용 환경을 위하여 일련의 컴포넌트 수정 개념, 특히 재정의 개념에 관하여 정의하고자 한다. 이는 향후 이를 지원할 수 있는 도구의 설계 및 구현을 위한 기반 개념이 된다.

1. 서론

소프트웨어 재사용의 개념은 유사한 부분의 반복적 작업을 줄이고, 이미 개발되어 있는 소프트웨어를 사용하므로써 생산성과 소프트웨어 품질 향상의 두 가지 목적을 달성할 수 있다. 이러한 소프트웨어 재사용의 개념은 개발 방법론의 발전과 더불어 재사용의 대상과 방법에서 여러 단계를 거쳐왔다.

재사용 대상의 형태에 따라서 여러 재사용 방법이 있으나, 특히 컴포넌트 기반 소프트웨어 재사용(Component Based Software Reuse)은 합성(Composition)에 의한 소프트웨어 개발 방법론으로서, 여러 컴포넌트가 함께 동작하도록 조립하는 과정이 곧 소프트웨어를 개발하는 과정이 된다. 그러므로 CBD(Component-Based Development)는 기존에 소프트웨어 개발 방법론이 가지고 있던 개별적인 모듈의 생성기법에서 조립과 통합의 기법이 사용되어야하고, 또한 애플리케이션 생성기 또는 자동 코드 생성기를 통한 개발 또는 재사용에서 모듈 통합 언어 또는 도메인 별 소프트웨어 아키텍쳐를 통한 소프트웨어 개발을 목표로 한다[1,2,3,4]. 이에 본 연구에서는 컴포넌트 재사용 환경을 위한 컴포넌트 수정개념을 정립하고자 하며 특히, 재정의(customization)개념을 정립하고자 한다. 이는 향후 이를 지원할 수 있는 도구의 설계 및 구현을 위한 기반 개념이 되도록 하고자 한다.

2. 컴포넌트 기반 소프트웨어 재사용 환경

컴포넌트를 재사용 한다는 것은 기존의 라이브러리 기반의 재사용과 프레임워크 기반의 재사용과는 다르기 때문에 새로운 형태의 재사용 환경이 필요하다. 그 특성은 다음과 같다.

① CBD의 특성 지원환경: 컴포넌트들끼리 표준화된 인터페이스의 결합을 통해 함께 동작할 수 있는 시스템으로의 통합을 지원하는 환경이 필요하다.

② 컴포넌트의 자체특성 지원환경 : 본 연구에서 관심을 갖는 재사용 대상인 컴포넌트는 소스 코드가

제공되지 않는 블랙박스 형태의 코드로 실행 가능한 독립성을 지니고 있다. 기존의 코드가 제공되는 화이트박스 형태에서의 코드의 수정이나 상속을 통한 코드의 재정의가 불가능하다. 그러므로 기존의 코드 기반의 재사용과는 다른 형태의 재사용 기법과 이를 지원하는 기술들이 요구된다.

③ 도메인 아키텍쳐 기반 재사용의 환경 필요성 : 블랙박스 형태의 컴포넌트가 모든 애플리케이션에 걸쳐 재사용이 가능한 범용 컴포넌트라 할지라도 컴포넌트는 재사용 가능한 범위(Context)를 갖는다. 이는 컴포넌트의 인터페이스에 의해 정의될 수 있는 부분이다. 즉, 제공되는 Provided & Required 인터페이스의 특성에 따라 다른 컴포넌트 또는 다른 애플리케이션과의 통합이 가능하다. 특정 도메인을 지원하는 컴포넌트의 경우라면 해당 도메인의 특성에 따라 정의된 인터페이스의 범위와 의미가 다를 수 있기 때문에 컴포넌트의 재사용이 효과적이기 위해서는 컴포넌트를 재사용할 수 있는 아키텍쳐 기반으로 재사용이 이루어져야 한다.

3. 컴포넌트 재정의 개념

3.1 정의

컴포넌트의 재정의는 프라퍼티의 변경을 의미한다. 이는 지금까지 클라이언트 컴포넌트에서도 이루어져 왔던 가장 기본적인 컴포넌트의 재정의 형식이다.

프라퍼티란 컴포넌트의 범위와 행위를 결정짓는 특성으로 컴포넌트를 이용하여 애플리케이션을 설계할 때 그 값을 변경할 수 있다. 컴포넌트는 애플리케이션 설계 시에 자신에 정의된 프라퍼티를 공개하도록 설계되어진다. 컴포넌트에 정의된 속성(Attribute)과 비교한다면, 속성은 컴포넌트에 정의되어야할 정적인 자료 저장소이다. 그러나 프라퍼티는 실행시의 컴포넌트의 구조나 행위의 형태를 결정짓는 특성이다. 이러한 프라퍼티로 정의될 수 있는 대상을 생각해 보면 다음과 같다.

① 속성의 종류 : 컴포넌트의 재정의 방식에 따라 컴포넌트가 저장하고 관리해야 하는 속성들의 종류가 달라지게 된다. 이는 컴포넌트를 사용하는 상황(Context)에 따라 달라 질 수 있다.

② 인터랙션(Interaction) : 컴포넌트들간의 또는 컴포넌트가 하나의 행위를 처리해 나가는 행위의 워크플로우(Workflow)와 컴포넌트에 정의된 연결 정보를 의미한다.

③ 행위 : 컴포넌트의 제약 조건을 만족하면서 컴포넌트에 정의된 인터페이스의 동작 방식이 달라 질 수 있다.

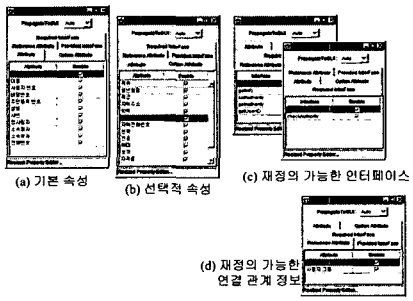
컴포넌트의 재정의는 정의된 프라퍼티에 의해 그 범위가 결정된다. 본 연구에서는 컴포넌트의 의미있는 재정의를 위해서는 그림1과 같은 내용의 프라퍼티 정의가 필요하다. 이는 본 연구에서 정의하는 컴포넌트 재정의 범위이기도 하다.

(a) 기본 속성(Attribute) : 컴포넌트가 가장 기본적인 행위를 수행하기 위해서는 반드시 정의되어야 하는 저장소를 의미한다. 이는 컴포넌트의 동작 중에 필요한 식별자나 기본 행위의 처리에 필요한 의미있는 자료들이 해당된다. 이들 속성이 누락될 경우 컴포넌트의 동작이 불가능함을 구분하는 것으로 재정의 과정에 포함되지 않는 컴포넌트의 기본 특성에 해당한다.

(b) 선택적 속성 : 컴포넌트의 재사용 범위에 따라 삭제할 수 있는 속성들이다. 이들 속성의 누락은 이들 속성을 활용하는 행위도 함께 삭제됨을 의미한다.

(c) 재정의 가능한 인터페이스 : 컴포넌트가 제공하는 제공 (Provided) 인터페이스와 요구(Required)인터페이스를 선택하여 정의할 수 있다.

(d) 재정의 가능한 연결 관계 정보 : 컴포넌트가 동작하기 위해 필요한 다른 컴포넌트에 대한 정보를 선택함으로서 컴포넌트의 동작 방식을 재정의 할 수 있다.



<그림 1> 서버 컴포넌트의 재정의 대상

3.2 컴포넌트 재정의를 위한 기반 개념

프라퍼티에서 가장 기본이 되는 것은 단순 프라퍼티(Simple Property)로 일반적인 프로그래밍 언어가

제공하는 타입을 갖는 프라퍼티이다. 프라퍼티는 그 값의 재정의를 위해서는 반드시 정의된 메소드를 통해서만 접근 가능하다. readable 프라퍼티의 경우는 getter 메소드를 제공하며 수정이 가능한 프라퍼티의 경우는 setter 메소드도 제공하여 그 값을 변경할 수 있도록 해야한다. 단순 프라퍼티와 동일한 특성을 지니지만, 정의할 수 있는 값의 종류가 참, 거짓만을 갖는 부울(Boolean) 프라퍼티가 존재한다. 부울 프라퍼티 역시 단순 프라퍼티와 동일한 방식으로 재정의 가능하다. 또한 값의 범위를 갖도록 정의하는 프라퍼티를 인덱스형 프라퍼티(Indexed Property)라고 정의한다. 이는 값의 범위를 갖고 있으며, 범위 안에 속한 각각의 값들은 인덱스가 부여되어 있어, 이들 프라퍼티를 읽거나 수정하기 위해서는 반드시 부여된 인덱스를 이용하여야 한다. 이러한 인덱스형 프라퍼티는 정의된 값들을 배열의 형태로 제공하는데, 이는 도메인 분석을 통해 컴포넌트의 속성이 가질 수 있는 값의 범위가 이미 정의된 경우에 사용할 수 있다. 또한 하나의 프라퍼티의 변경으로 인하여 다른 컴포넌트에 그 변화의 영향이 반영되어야 하는 경우가 발생할 수 있는데 이런 프라퍼티를 Bound 프라퍼티라고 정의한다. 그리고 컴포넌트의 속성을 재정의 할 때 아무런 제약 조건 없이 어떤 값으로든지 수정가능한 경우도 있지만, 도메인의 특성과 비즈니스 전략에 따라서 어떤 조건(Invariant)를 만족하는 경우에만 변경을 허용해야 하는 경우가 있다. 여기서는 이를 Constrained 프라퍼티라고 부른다.

3.3 재정의를 위한 프라퍼티 및 메소드 기법

컴포넌트를 재정의하기 위해서는 컴포넌트에 정의된 프라퍼티를 알아내야 한다. 이러한 기법을 Introspection이라고 한다. Java 컴포넌트의 경우는 별도의 인터페이스 명세 없이 코드 분석을 통해 필요한 정보를 추출할 수 있도록 제공하고 있다[5,6,7]. 먼저 Java 컴포넌트 분석기를 위한 규칙을 정의하고 다른 컴포넌트를 위한 별도의 명세 규칙을 정의한다 단순 프라퍼티의 경우는 아래의 규칙을 갖는다.

<규칙 1>

```
public <PropertyType> get<PropertyName> ();
public void set<PropertyName> (<PropertyType> a);
```

반드시 get<PropertyName>과 일치되는 set<PropertyName>를 갖고 있어야 하며 그들의 <PropertyType>은 동일해야 한다. 이로써 지원 도구에서 분석할 프라퍼티의 이름은 <PropertyName>이 된다. 부울(Boolean) 프라퍼티의 경우는 아래의 규칙을 갖는다.

<규칙 2>

```
public void add< EventListenerType>(<EventListenerType> a)
public void remove< EventListenerType>(<EventListenerType> a)
```

인덱스형 프라퍼티의 규칙은 다음과 같다.

<규칙 3>

```
public boolean is< PropertyName>();
```

컴포넌트에 정의되는 이벤트에 대한 규칙은 다음과 같다.

<규칙 4>

```
public < PropertyElement> get< PropertyName>(int a);
public void set< PropertyName>(int a, < PropertyElement> b);
```

다른 메소드에 대한 엄격한 규칙은 정의되어 있지 않다. 단지 BeanInfo를 통해 프라퍼티, 이벤트, 메소드에 대한 구체적인 정보를 얻을 수 있다. BeanInfo 클래스는 정의된 컴포넌트 이름에 BeanInfo를 연결하여 만들어진다. 컴포넌트의 BeanInfo 클래스는 컴포넌트의 행위에 대한 명세로 사용할 수 있는데, 예를 들어 getEventSetDescriptors의 메소드를 통해서 EventSetDescriptor를 담고 있는 배열을 얻어 낼 수 있다. 이 EventSetDescriptor배열은 컴포넌트에 정의된 이벤트에 대한 정보를 정의하고 있다. 그러나 이러한 정보는 EJB에 국한하여 추출할 수 있는 것으로 보다 범용적 재정의를 위해서 이와 별도로 컴포넌트의 인터페이스를 정의하기 위한 명세를 BNF 형식으로 정의 할 필요가 있을 것이다.

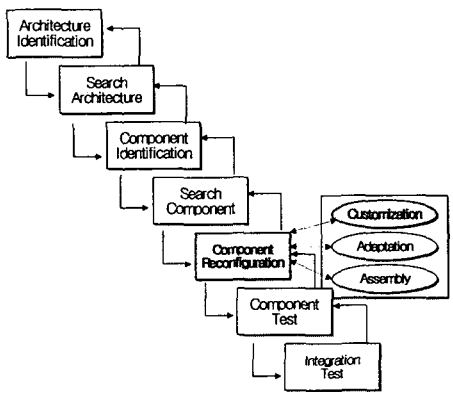
3.4 프라퍼티 및 인터페이스 재정의를 위한 참조내용

이는 향후 프라퍼티 재정의를 위한 도구를 구축할 때 미리 제공하는 여러 컴포넌트에 대한 프라퍼티 및 인터페이스 재정의 참조내용에 관한 것으로 다음과 같은 예시를 들 수 있다.

클래스 이름	java.beans.PropertyVetoException
상위 클래스	java.lang.Exception
메소드 이름	설 명
public PropertyVetoException (String mess, PropertyChangeEvent evt)	mess : 설명문 evt : 변경을 반대하는 이유를 설명하는 PropertyChangeEvent

4. 결론 및 향후과제

본 연구에서는 기존의 코드 기반 또는 문서 기반의 컴포넌트보다는 독립적으로 실행 가능한 블랙박스 개념의 컴포넌트를 기반으로 하여, 소프트웨어 재사용을 지원하는 환경을 위하여 컴포넌트 수정을 위한 컴포넌트 재정의 개념을 정립하고자 했다. 이는 컴포넌트의 재정의 기술을 축적한 후 새로운 어플리케이션 개발을 효과적으로 지원하기 위하여 재정의 지원도구를 통한 어플리케이션 개발과정의 자동화를 지향하고자하는데 목적이 있다. 이를 위해서는 컴포넌트가 어떤 이유로든 미리 정의된 인터페이스나 행위를 변경해야할 경우에 필요로 하는 컴포넌트 수정(Adaptation)과 독립적으로 개발된 컴포넌트를 조립하여 하나의 어플리케이션을 만들어가는 조립(Assembly)에 대한 기법 정의가 추가로 이어져야 할 것이다. 이를 통해 컴포넌트 재사용을 위한 환경적인 요소를 정립하고 이를 지원해줄 수 있는 Activity를 바탕으로 하는 프로세스 모델을 정의할 수 있을 것이다. 재정의를 중심으로하는 개략모델을 <그림2>와 같이 예측해 볼 수 있으며, 이러한 개념을 바탕으로 재정의 지원도구의 설계와 구현이 가능해지리라 본다. 이에 어플리케이션 개발자는 컴포넌트를 선택하여 기본 행위, 구조등을 이해하고 재정의 부분을 식별한 후 컴포넌트 재정의 도구를 통해 속성을 변경하거나 추가될 부분을 명세하여 새로운 컴포넌트를 정의하는 과정을 반복하므로 새로운 어플리케이션을 구성하게 된다.



<그림2> 컴포넌트 기반 재사용 시스템을 위한 프로세스 개략모델

참고문헌

1. Mettala,E,Graham M,"The Domain-Specific Software Architecture Program", Techinical Reports CMU/SEI-92-SR-9, Carnegie Mellon University, June, 1992

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

4. Microsoft Corp,"The Component Object Model : Technical Overview", Dr. Dobbs Journal, December 1994

5. James Gosling, Bill Joy and Guy Steele. The Java language Specification. Addison-Wesley, 1996.

6. Tim Lindholm and Frank Yellin. The Java Virtual Machine Specification, Addison-Wesley, September 1996.

7. Thomas C. Valesky, et al, Enterprise Javabeans : Developing Component-Based Distributed Applications, Addison-Wesley, 1999