CBD를 위한 기존 클래스의 JavaBean으로의 변환에 관한 연구

김행곤*, 김지영*, 김병준**
*, **대구가톨릭대학교
*{hangkon, jykim}@cataegu.ac.kr, **windjun@kebi.com

A Study for Transformation from legacy class to JavaBeans for Component Based Development

Heang-Kon Kim, Ji Young Kim, Byung Jun Kim
Dept. of Computer Engineering, Catholic University of Daegu

요 약

컴포넌트기반의 개발은 오랫동안 소프트웨어 개발의 관건이었던 재사용의 초점을 코드나 클래스 라이브러리보다 발전된 형태인 컴포넌트에 초점을 맞추고 있으며, 이는 보다 진보된 형태의 재사용이라 볼 수 있다. 하지만 재사용방법이 기존 어플리케이션에서의 재사용이라기 보다는 또 다시 새로운 형태로의 재사용 컴포넌트를 개발하여 구축하는 경우가 허다하며, 또한 대부분 소규모 컴포넌트 개발에 한정되어 있다. 자바 어플리케이션의 경우 자바 언어 기반의 컴포넌트 모델이 존재하지만, 소규모의 새로운 재사용 단위나 제한된 GUI 컴포넌트 개발에만 머무르고 있다. 따라서, 컴포넌트를 기반으로 한 개발임에도 불구하고, 그 장점을 충분히 발휘하지 못하거나 부대적인 비용, 노력을 낭비하는 경우가 종종 있으며, 또한 특정 도메인 컴포넌트에서만 두드러진다는 것이다.

따라서, 본 논문에서는 기존의 자바 어플리케이션을 분석하여 컴포넌트화 할 수 있는 부분을 확장하여 식별하고, 재사용단위로서 비즈니스 로직의 부분적인 수용을 통해 기존 어플리케이션을 자바빈으로 변환하는 기법을 제시하고자 한다.

1. 서 론

컴포넌트의 조합에 의해 소프트웨어를 개발하는 CBD는 소프트웨어 개발의 적시성, 생산성 향상, 품질 향상 등 기존의 개발방법론보다 진보된 방법론임에는 틀림이 없다. 이미 국내에서도 KCSC(한국SW컴포넌트 컨소시엄), N3소프트나 컴포넌트뱅크등이 국내 컴포넌트 시장 선점에 활발히 움직이고 있다. 이러한 변화는 CBD방법론이 가진 뛰어난 재사용성에 기인한 것이다[1]. 일반적으로, 이러한 재사용 가능한 컴포넌트는 여러 가지 환경과 상황에서 일관적인 동작을 수행하여야 하며, 컨테이너 또는 사용자의 빌더에서 적용되어 사용할 수 있도록 재사용을 지원할 수 있어야 한다. 또한 CBD의 궁극적인 목표인 재사용의 효율적인 이용은 그 재사용 단위인 컴포넌트의 정규화된 실용적 접근에 의존한다고 볼 수 있다. 이러한 관점에서 객체지향언어이자 플랫폼 독립적인 자바 언어의 컴포넌트 모델인 자바빈즈은 아주 적합한 컴포넌트 모델이다. 또한, 자바빈 컴포넌트는 단일하고 고성능의 API를 제공하며, 단순하다[2]. 하지만 기존의 자바빈 컴포넌트의 주 활용범위는 RAD도구나 IDE에서 UI 레이아웃을 지원하거나 아주 미비한 재사용을 지원하는 등의 일부 영역에 제한된 재사용에 불과했다. 그러나 이러한 제한된 재사용은 기존 어플리케이션에서의 비즈니스 로직자체의 패키징이나 GUI 컴포넌트와 직접 연관되는 도메인 로직의 부분적인 수용으로 재사용의 효과를 넓힐수 있다.

따라서, 본 논문에서는 자바기반의 어플리케이션을 컴포넌트의 행위적인 측면에서 분석하여, 자바빈의 설계 서식에 따라 컴포넌트화의 범위를 확대하고, 재사용단위로서 도메인 로직의 부분적인 수용을 통해 기존 어플리케이션을 자바빈으로 변환하는 기법을 제시하고자 한다.

2. 관련 연구

2.1 JavaBeans

자바빈즈는 자바 언어를 위한 소프트웨어 컴포넌트 모델이며 빌더도구에서 시각적으로 조작 가능하다. 또한 서드 파티가 생산, 유통하여 최종 사용자가 재사용할 수 있도록 고안되었으며 Sun사에서 제안한 플랫폼 중립적인 소프트웨어 컴포넌트 아키텍처를 기반으로 한다[3].

자바빈즈는 <표 1>에 명시된 명명규칙에 따라 작성됨으로써, 코어 리플렉스(Core Reflection) API를 통해 실행가능하며, 인트로스펙터(Introspector) 클래스를 통해 검사될 수 있다. 인트로스펙터 클래스는 빈이 가지고 있는 클래스, 인터페이스, 메소드, 그리고 아규먼트를 찾기 위해 코어 리플렉션을 사용한다.

<표 1> 리플렉션을 위한 디자인패턴

Design patterns for Properties
Simple properties
public *<PropertyType>* get*<PropertyName>*();
public void set*<PropertyName>*(*<PropertyType>* a);
Boolean properties
public boolean is*<PropertyName>*();
public void set*<PropertyName>*(boolean *<PropertyName>*);
Indexed Properties
public *<PropertyElement>* get*<PropertyName>*(int a);
public void set*<propertyName>*(int a, *<propertyElement>* b);
Bound Properties
Constrained Properties
Design Patterns for Events
public void add*<EventListenerType>*(*<EventListenerType>* a)
public void remove*<EventListenerType>*(*<EventListenerType>* a)

또한, 자바 빈즈의 구현에는 직렬화(Serilization), 명명 관례, 설계 관례 등 몇 가지 세부사항을 지침해 두고 있다. 자바빈즈는 최종사용자의 IDE에 배치되어 요구되는 동작이 수행되고, 재사용이 이루어질 수 있도록 파악되어져야 한다. 따라서 적절한 프로그래밍 패턴이 요구되며, 이러한 설계 규칙에 의해 작성된 빈은 코어 리플렉션 API를 사용하는 인트로스펙션 클래스에 의해 빈의 API를 프로그래밍적으로 자동 분류될 수 있다. 또한 속성, 이벤트, 메소드의 디자인 패턴에 따른 명명으로 사용자 및 자바빈 기반의 IDE에서 자바빈즈는 인식되고 조작되어 질 수 있다[2][3].

2.2 컴포넌트 추출 및 변환

CBD는 고품질의 컴포넌트 구축과 컴포넌트 아키텍처 또는 컴포넌트 프레임웍의 용이한 구성으로 완성된다. 고품질의 컴포넌트 구축에는 어플리케이션 개발과 상이한 목적으로 새로운 컴포넌트를 구축하는 관점과 이미 기업환경이나 최종사용자들에게 그 품질을 인정받은 기존 어플리케이션으로부터 가치있는 비즈니스 로직을 추출하여 컴포넌트화하는 관점, 두 가지를 들 수 있다. 기존의 어플리케이션으로부터 컴포넌트를 추출하거나 재구축하는 경우 프로그램 품질의 신뢰도의 향상과 개발 사이클의 감소, 향후 재사용에 따른 부가 가치 등 여러 가지 이점을 가질 수 있다[4][5].

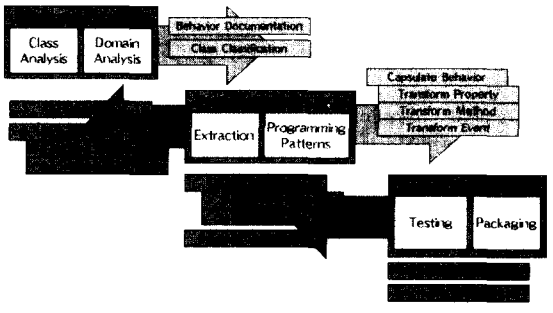
현재 CBD를 도입, 구현하는 Netron Process나 E-Business 구축 솔루션들은 이러한 기존 어플리케이션에서의 컴포넌트 추출을 적극 수용하고 있다.

3. 자바빈즈로의 변환

기존 어플리케이션으로부터 추출된 컴포넌트가 배치될 환경에서 무리 없이 요구되는 도메인 로직을 수행하기 위해서 가장 중요한 것은 구축된 컴포넌트 모델과 배치된 컴포넌트 아키텍처와의 관계이다. 본 논문에서 제안하는 컴포넌트 변환 프로세스는 자바 기반의 어플리케이션으로부터 자바의 컴포넌트 모델인 자바빈즈로 추출, 구축함으로써 컴포넌트를 설계, 구현, 사용함에 있어 잠재적인 오류를 줄여준다.

3.1 JavaBeans 변환 프로세스

자바빈즈 컴포넌트의 구축은 기존 자바 어플리케이션을 분석하고, 소스 코드를 통해 재사용 단위를 추출해 내어 도메인 관점에서 재사용 가능한 컴포넌트 모델로의 변환을 목적으로 한다. 여기엔 객체, 클래스, 메소드, 데이터, 컴포넌트 인터페이스를 포함하는 설계 정보들의 추출을 포함한다. 본 논문에서 제안하는 프로세스(그림 1)는 자바 빈즈로의 변환을 목적으로 자바로 작성된 기존 어플리케이션을 전제로 한다.



(그림 1) 자바빈즈 변환 프로세스

<표 2> 클래스 형태에 따른 분류

구분	특징
UI Class (분류1)	UI를 구성하는 속성들과 메소드로 이루어진 클래스, 자바빈즈로 변환 가능하거나 자바빈즈의 형태로 담겨진 클래스
Pure Domain Class (분류2)	순수하게 도메인 로직을 수행하는 클래스, UI Class와 상호작용이 있을수 있으며, Invisibel 빈의 성격을 가짐
GUI Related Domain Class (분류3)	UI를 구성하는 속성과 메소드에 도메인 로직을 포함하는 클래스

① 첫 번째 단계, **어플리케이션 분석**에서 목적 도메인의 관점에서 분류하여 UI widget이나 순수 도메인 로직 클래스, UI와 연관되어 도메인 로직을 수행하는 클래스를 추출해내며 어플리케이션의 행위적인 관점에서의 식별과 문서화 과정을 가진다.

<표 2>는 어플리케이션 분석 단계에서 추출되는 클래스의 분류를 나타낸다. 일반적으로 UI를 구성하는 클래스(분류1)는 그 자체로 빈즈가 될 수 있으나 그 규모가 너무 작고 이미 대부분의 IDE에서 사용이 용이하다. 두 번째 순수 도메인 로직을 수행하는 클래스(분류2)는 Invisible 빈의 성격을 띠며 (분류1) 클래스들에 메소드를 호출하거나, 이벤트를 발생시키고, 상태를 저장시킨다. 하지만, 도메인 로직의 재사용자체가 비즈니스 형태의 변화에 따라 민감하므로 컴포넌트화의 의미가 노력에 비해 적은 효과를 가진다. 따라서 본 논문에서는 (분류3)의 클래스들에 초점을 두고 있다. (분류3)은 UI를 구성하는 클래스에 부분적인 도메인 로직을 담고 있는 형태로써 적절한 클래스 분리를 통해 보편적인 컴포넌트 형태로 구축될 수 있다.

② 두 번째 과정인 **변환**은 본 프로세스의 코어로써 빈즈로의 변환 과정을 가진다. 자바빈즈의 프로그래밍 패턴에 입각한 추출 클래스들의 변환을 위해 속성, 메소드, 이벤트의 행위적인 캡슐화가 이루어지며 필요에 따라 추가되거나 삭제되는 요소들이 있을수 있다.

③ 마지막 단계인 **커스터마이징**은 변환된 자바 빈즈를 그 목적에 맞게 IDE나 최종사용자가 사용할 수 있도록 지원하는 단계이다. 자바빈즈의 특성에 따라 커스텀 인스펙터가 작성되어질 수 있으며, 테스팅, 배포를 위한 패키징 과정과 자바빈즈에 대한 문서화를 포함한다.

3.2 변환을 위한 속성, 이벤트, 메소드 제메치

변환에 있어서의 핵심적인 내용은 본래의 자바 어플리케이션을 자바빈즈의 속성 접근 서식, 이벤트 소스 서식 등 개발자나

<표 3> 자바빈즈 설계 서식

속성 접근 서식	
Simple	get/set*PropertyName*
Boolean	is/set*PropertyName*
Indexed	add(<IndexName/Type>)

Bound	get/is/set*PropertyName*
	addPropertyChangeListener();
	removePropertyChangeListener();

Constrained	public void set<PropertyName>
	(<PropertyType><PropertyName>) throws
	PropertyVetoabieException;
	addVetoableChangeListener();
	removeVetoableChangeListener();

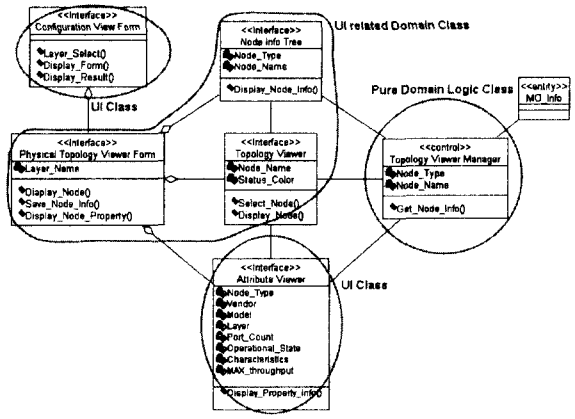
PropertyDescriptor by Introspector	
이벤트 서식	
multicast	public void add<EventListenerType> (<EventListenType><EventListenerName>); public void remove<EventListenerType>
unicast	public void add<EventListenerType> (<EventListenerType><EventListenername>) throws java.util.TooManyListenerException; public void remove<EventListenerType>
EventSetDescriptor by Introspector	

IDE가 접근하여 인식할 수 있도록 자바빈즈 프로그래밍 패턴을 바꾸어주어야 하며, 이러한 과정에서 메소드가 추가되거나 자바빈즈 프로그래밍 패턴에 적합하지 않은 메소드의 변형 또는 이벤트 어댑터의 추가, 제거가 있을 수 있다. <표 3>는 자바빈즈 변환에 있어서 기본적으로 추가되어야 할 서식을 나타낸다.

4. 자바 빈즈 컴포넌트 추출 사례

4.1 행위적 관점의 클래스 분류

본 논문에서는 기존에 자바로 작성된 NMS시스템에서 망관리를 수행하는 어플리케이션에 변환 프로세스를 적용하여 행위적인 관점에서 속성, 메소드, 이벤트를 분류하였다. 망관리 사용자 인터페이스는 (그림 4)와 같이 GUI 클래스, 도메인 로직과 UI표현을 동시에 수행하는 클래스, 마지막으로 순수도메인 로직만 수행하는 Control 클래스로 분류된다.



(그림 2) 클래스 분류

그리고 자바 빈즈 변환프로세스 첫 번째 단계의 "Class Classification"의 결과물로 다음의 <표 4>와 같이 정리하였다.

<표 4> Class Classification

구분	클래스	연관 클래스
UI Class (분류1)	PhysicalLayerPanel DrawNode, DrawLink	DrawGridScrollPane
Pure Domain Class (분류2)	DrawObject DrawObjectLinkedList Module	DrawNode, DrawLink
GUI Related Domain Class (분류3)	DrawGridScrollPane SetNodeAttributeDialog SetLinkAttributeDialog	DrawGridScrollPane

4.2 속성, 메소드의 캡슐화

분류된 각각의 클래스들은 앞의 <표 3>의 자바빈즈의 설계 관례인 디자인 패턴과 설계 서식을 적용한다. 자바빈즈 변환시 속성, 이벤트, 메소드를 중심으로 기존 클래스에서 변경과 변환된 속성, 메소드를 위한 어댑터 클래스가 추가된다<표 5>.

추가된 서식은 클래스의 이름과 기본 클래스 그리고 구현된 인터페이스를 인용한다. 또한, 속성 리스너 인터페이스의 구현과 이들 리스너에 대한 add/remove 서식도 변환될 클래스의 특성에 따라 포함될 수 있다.

<표 5> 코드 변환

Legacy code	속성	private JTextArea descriptionTextArea;
	생성자	descriptionTextArea = new JTextArea();
	속성 접근 서식	descriptionTextArea.setText (objCurrentNode.m_description);
Transformed code	속성	private JTextArea descriptionTextArea; private static final String DES_TEXT_PROPERTY = "Description"; private String descriotionText; private PropertyChangeSupport propertySupport;
	생성자	descriptionTextArea = new JTextArea(); propertySupportSupport = new PropertyChangeSupport(this);
	속성 접근 서식	public String getDescription(){ return descriptionText; } public void setDescription(String value) { String oldValue = descriptionText; desciptionTest = value; } propertySupport.firePropertyChange (DES_TEXT_PROPERTY, oldValue, descriptionText) }

이러한 코드변환의 결과로 재작성된 자바빈즈는 코어 리플렉션에 의해 인식되어 IDE나 사용자에게 자신의 속성과 메소드에 대한 정보를 제공할 수 있다.

5. 결론 및 향후 연구

컴포넌트 저장소가 속속들이 구축되어지고 있는 현 시점에서 기존의 자바 어플리케이션으로부터 자바빈즈를 구축해나가는 방법은 보다 경쟁적이라 할 수 있겠다.

본 논문에서는 이를 지원하기 위해 자바 기반의 어플리케이션으로부터 자바의 컴포넌트 모델인 자바빈즈로의 변환 기법을 제시하였으며, 기존의 NMS 어플리케이션에 적용, 컴포넌트 추출 및 코드 매핑을 구현해 보았다. 그 결과로 레거시 어플리케이션이 대규모 단위일수록 변환에서 코드 매핑의 복잡성이 증가하고, 재사용성이 떨어지는 단점이 예상되지만, 변환 알고리즘의 연구와 자동화로 보완할 수 있다. 따라서, 향후 연구로는 변환 프로세스의 정형화와 이를 지원하는 자동화된 도구의 개발이 이루어져야 할 것이다.

[참고 문헌]

[1] 정보처리학회지, "컴포넌트 소프트웨어", 정보처리학회지 Vol. 7, No. 4, 2000.

[2] Sun Microsystems, Inc., JavaBeans Specfication, version 1.01, 1997.

[3] Dan Brookshier, 자바빈즈 개발자 레퍼런스, 대림출판사, 1997.

[4] Clemens Szyperski, Component Software, Addison-Wesley, 1998.

[5] Netron Process, http://www.netron.com/bubbles/build.

[6] Sun Microsystems, Inc, How to be a Good Bean, 1997.

[7] Ivor Horton, Beginnig Java2, 정보문화사, 2001.