JNI를 이용한 레거시 어플리케이션의 컴포넌트 Wrapper 설계

백수진*, 송영재**
*경희대학교 컴퓨터공학과
e-mail : croso@cvs2.khu.ac.kr

Design of Component Wrapper from legacy application using Java Native Interface

Su-Jin Baek*, Young-Jae Song**
Dept of Computer Engineering, Kyung-Hee University

요 약

컴포넌트 기반의 소프트웨어를 개발하는 데 있어서 대부분의 방법론은 앞으로 구축할 새로운 시스템의 재사용성 확보에 치중할 뿐, 기존의 레거시 시스템 자원을 재사용하여 새로운 시스템을 구축하는 실용적인 재공학 방법을 지원하지 못하고 있다. 또한, 기존의 레거시 시스템을 컴포넌트화하는 방법들도 소프트웨어들의 규모가 방대해지고 복잡해짐으로써 시간적, 비용적 측면에서 많은 시간과 노력이 요구된다.

본 논문에서는 레거시 어플리케이션을 재공학 프로세스를 적용하여 컴포넌트화하기 원하는 메소드를 추출하고, WDL 정의 및 JNI를 연계하도록 하는 래퍼를 생성하여 자바빈즈 컴포넌트화함으로써 범위를 최대화하고, 재사용성을 극대화할 수 있는 방안을 제시한다.

1. 서론

컴포넌트 개발 방법론이 일반화되면서 기업에서는 기존 소프트웨어의 재사용과 유지보수의 필요성을 느끼게 되었다. 일반적인 컴포넌트 기반의 소프트웨어 개발에는 컴포넌트를 자체적으로 개발하거나 제3자가 개발한 컴포넌트를 구입하여 사용하게 된다. 그러나 이러한 경우, 많은 자원 낭비뿐만 아니라 최악의 경우 재사용을 고려한 소프트웨어를 처음부터 개발해야 한다는 문제점을 가진다. 또한, 기존의 방법론들은 새로운 시스템의 재사용성 확보에 치중할 뿐, 실질적으로 기존의 레거시 시스템의 자원을 활용하여 새로운 시스템으로 재개발하거나 재사용 가능한 컴포넌트를 추출하는 방법 등은 명확히 제시되지 못하고 있다.[1]

레거시를 컴포넌트화하는 여러 방법 중 레거시 전이(Legacy migration)는 기존 플랫폼에서 새로운 플랫폼으로 데이터를 이동하는 방법으로 주로 사용되고, 단일화된 기존 시스템을 래퍼와 인터페이스를 통하여 사용되는 레거시 래핑(Legacy wrapping)은 많이 사용되나 블랙박스로 제공됨으로 사용자가 쉽게 대체할 수 없는 단점을 가지고 있다. 레거시 재구조화(Legacy restructuring)은 하나의 기존 시스템의 응집도를 높이고, 결합도를 낮추어 컴포넌트화하는 방법이나 최근의 많은 소프트웨어들이 규모가 방대하고 복잡하여 시간적, 비용적 측면에서 많은 시간과 노력을 요구한다. 또한, 기존의 자바빈즈 컴포넌트의 주 활용범위는 RAD(Rapid Application Development) 도구나 IDE(Integrated Development Environment)에서 사용자 인터페이스 정도의 미비한 재사용에 불과하다.[2]

따라서, 본 논문에서는 레거시 어플리케이션을 재공학 프로세스를 적용하여 컴포넌트화하기 원하는 메소드를 추출하도록 하였고, 기존 레거시 언어로 작성된 부분을 WDL로 정의하고, 자바네이티브 인터페이스(JNI)를 사용하여 래핑함으로써 자바빈즈 컴포넌트로 사용할 수 있는 방법을 제안하였다.

2. 관련연구

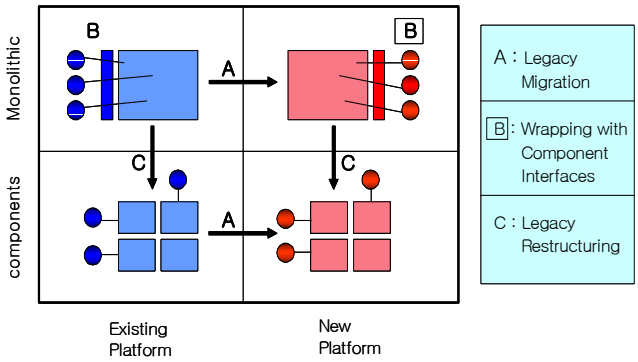
본 장에서는 기존의 컴포넌트화를 위한 분석 방법과 자바빈즈 디자인 패턴에 대해 살펴보고, 레거시 시스템을 컴포넌트화하는 기존 방법에 대해 기술한다.

2.1 기존의 컴포넌트화를 위한 분석 방법

기존의 레거시 시스템의 재사용을 지원하는 재공학

방법으로는 UIRich System Redevelopment Methodology (USRM), Mission Oriented Architecture Legacy Evolution(MORALE) 등을 들 수 있다. USRM의 특징은 계획, 분석, 포지셔닝, 변환의 4단계로 구성되었으며 UI, 데이터, 기능 위주의 재공학 접근 방식을 제공한다. MORALE은 분석, 설계, UI 진화의 3단계로 구성되었으며 UI를 중심으로 한 재공학 접근 방식을 취하고 있다. 각 방법론들은 역공학을 통하여 기존 레거시 시스템의 자원을 추출 가능하다는 전제하에 구성되어 있다.[3]

2.2 레거시 시스템을 컴포넌트화 하는 기법



< 그림 1 레거시 시스템을 컴포넌트화하는 방법 >

레거시 시스템을 컴포넌트화하는 방법에는 <그림 1>과 같이 기존 플랫폼에서 새로운 플랫폼으로 데이터를 이동하는 방법으로 주로 사용되는 이전(Legacy migration) 방법과 단일화된 기존 시스템을 래퍼와 인터페이스를 통하여 사용되는 래핑(Legacy wrapping) 방법, 하나의 기존 시스템의 응집도를 높이고 결합도를 낮추어 컴포넌트화하기 위한 재구조화(Legacy restructuring) 방법이 있다.[1]

2.3 자바빈즈 디자인 패턴

자바빈즈(Javabeans)란 자바 언어로 개발된 포터블(portable)하고, 플랫폼 독립적인 컴포넌트 모델이다. 자바빈은 객체지향적인 소프트웨어 컴포넌트를 만들기 위한 아키텍처이고, 썬 마이크로시스템즈에서 표준으로 제시한 일련의 자바 API를 포함하는 자바의 컴포넌트 모델이다. 자바빈즈는 <표 1>에 명시된 명명규칙에 따라 작성됨으로써 코어 리플렉션(Core Reflection) API를 통해 실행 가능하다. <표 1>은 자바빈즈 변환에 있어서 기본적으로 추가되어야 할 서식을 나타내며, 이에 입각한 코드 매핑이 이루어진다. 이러한 디자인 패턴은 클래스와 메소드 명칭, 클래스 상속, 인터페이스의 구현, 메소드 인자 유형, 그리고 부분적 혹은 전체적으로 클래스나 메소드의 사용을 인지하기 위해 사용될 수 있는 예외들의 특징이다.

속성 접근 디자인 패턴
Simple 프로퍼티
public <PropertyType> get<PropertyName>(); public void set<PropertyName>(<PropertyType> a);
Boolean 프로퍼티
public boolean is<PropertyName>(); public void set<PropertyName>(boolean <PropertyName>);
Indexed 프로퍼티
public <PropertyElement> get<PropertyName>(int a); public void set<PropertyName>(int a, <PropertyElement> b);
Bound 프로퍼티
public void addPropertyChangeListener(); public void removePropertyChangeListener();
Constrained 프로퍼티

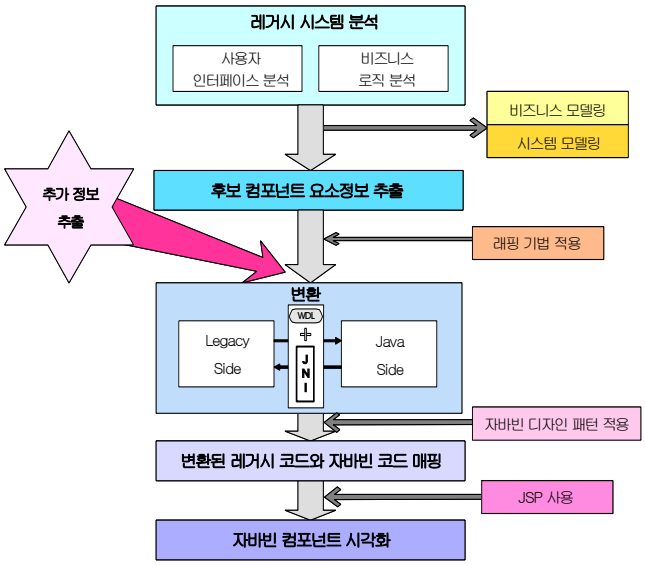
public void set<PropertyName>(<PropertyType><PropertyName>) throws PropertyVetoableException;
addVetoableChangeListener();
removerVetoableChangeListener();

이벤트를 위한 디자인 패턴
public void add<EventListenerType>(<EventListenerType>a) public void remove<EventListenerType>(<EventListenerType>a)

< 표 1 자바빈즈 디자인 패턴 >

3. 레거시 어플리케이션을 컴포넌트화하기 위한 설계

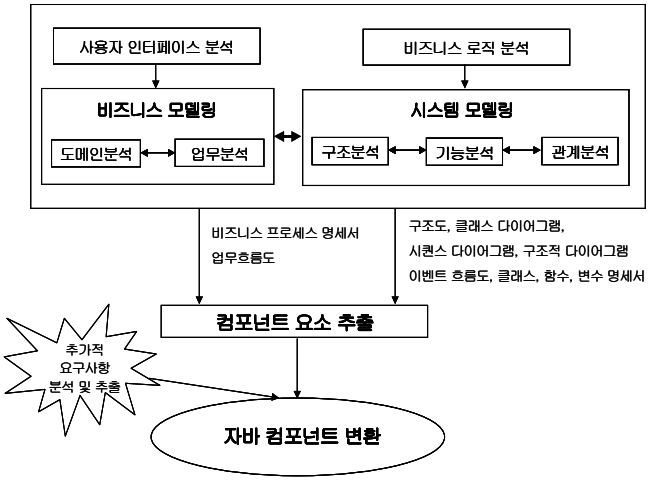
본 논문에서는 <그림 2>에서 보여지는 전체 프로세스에 대해 분석, 추출, 변환, 매핑, 시각화를 거쳐 레거시 어플리케이션을 자바빈즈 컴포넌트로 변환하게 된다. 이 장에서는 제시한 프로세스에 따라 레거시 언어로 작성되어 있는 개인정보관리시스템을 동기화하여 사용가능한 자바빈즈 컴포넌트로 변환하는 과정과 기존의 바



< 그림 2 전체 프로세스 >

이너리 파일 형태의 자바빈즈 컴포넌트의 명세화 과정을 사례를 중심으로 언급한다.

3.1 레거시 어플리케이션 분석



< 그림 3 레거시 어플리케이션에 대한 분석 프로세스>

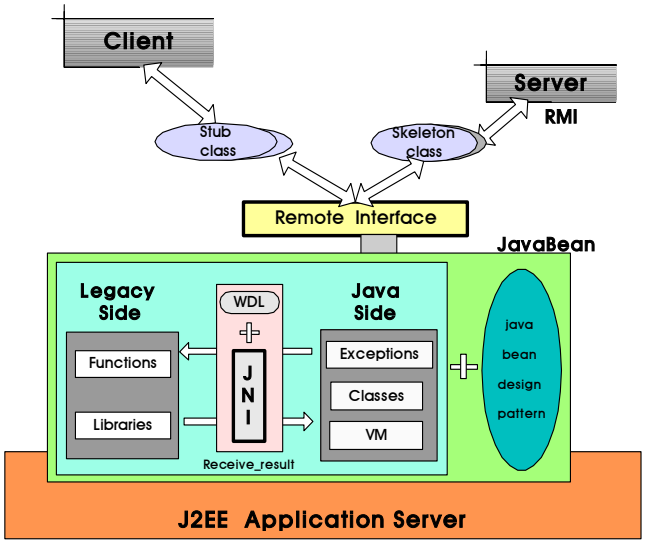
레거시 어플리케이션을 자바빈즈 컴포넌트로 변환하기 위해 먼저 기존의 레거시 어플리케이션에 대한 분석이 이루어져야한다. 레거시 어플리케이션은 대개 명세화되어 있지 않기 때문에 사용자 인터페이스, 비즈니스 로직으로 나누어 분석을 하게 된다. 이 단계에서는 이미 존재해 있는 컴포넌트로 변화될 대상 프로그램으로부터 클래스, 메소드, 이벤트 프로퍼티 등의 정보를 분석함으로써 비즈니스 모델링과 시스템 모델링을 통해 컴포넌트 요소를 추출하게 된다.

3.2 후보 컴포넌트 요소정보 및 추가정보 추출

레거시 어플리케이션 분석과정을 거친 후 생성되는 모델링들을 가지고 컴포넌트 요소에 대한 정보를 추출하게 된다. 이 부분에서 개인정보시스템에 관한 사용자 인터페이스에 대한 부분보다는 비즈니스 로직 부분에 치중을 두어 설명한다. 레거시 어플리케이션 자체의 추출된 정보만을 가지고 자바빈즈 컴포넌트로 변환하기에 부족함으로 이를 위해서 추가 정보 추출단계가 필요하다. 이 단계에서는 먼저 자바네이티브 인터페이스를 사용할 때의 문자열에 관한 문제에 대한 정보 및 자바빈즈 특성을 적용시키고, 동기화가 가능하도록 구현부분에서 자바빈즈 디자인 패턴을 통해 프로그램을 구현한다.

3.3 JNI를 사용한 자바빈즈 컴포넌트로의 변환

추출된 정보 요소인 메소드는 C++로 작성되어 있기 때문에 자바빈즈 컴포넌트로 변환하기 위해서 언어에 대한 문제를 해결해야한다. 이를 위해 <그림 4>에서 제안한 변환 구조와 같이 자바네이티브 인터페이스(Java Native Interface)와 WDL(Wrapper Definition Language)을 사용하여 C++로 작성된 시



< 그림 4 JNI를 사용한 자바빈즈 컴포넌트 변환 구조 >

스템 요소들을 자바 코드와 연계 가능하도록 래퍼를 생성하고, 자바빈즈 디자인 패턴을 적용한다.

1) WDL 파일 생성

WDL은 레거시 어플리케이션과 자바로 작성된 파일과의 연계를 위한 언어로서 분석과정을 통해 얻어지는 정보를 가지고 래퍼 메소드 명시부, 래퍼 속성 명시부, 래퍼 연결자 명시부로 정의한다.

2) JNI를 사용한 래퍼 생성

생성된 WDL 파일을 가지고 자바네이티브 인터페이스를 적용하여 래퍼를 생성하게 된다. 래퍼 메소드 명시부에서 정의된 메소드 명은 래퍼 연결자 명시부에 선언된 자바 파일의 메소드에서 호출하여 사용된다.

3) 자바빈즈 디자인 패턴 적용

WDL을 통해서 생성된 래퍼에 대한 정보로만 자바빈즈 컴포넌트로서의 기능을 수행할 수 없다. 따라서, 자바빈즈가 가져야 할 필수 항목의 조건을 만족하는지를 살펴보고, 자바빈즈의 필수 항목 조건을 갖춘 자바빈즈 디자인 패턴에 적용시켜 프로퍼티를

속성 접근 서식	

Simple	public int[] getEntryID(int n);
	pubic long getEntryNumber();
	public long getPropertyNumber(int id);
	public long setTagToLong(java.lang.String name);

Indexed	public int setItemField(int id, int no); public void setOneProp(long name, java.lang.String value, int[] newentryID);
Bound	public void dispatch(DatabaseListener listener);
이벤트 서식	

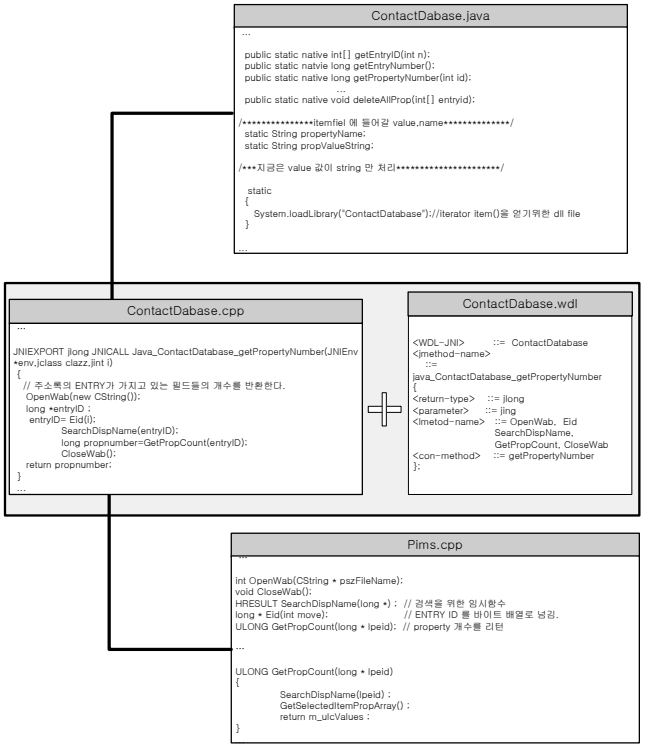
public void databaseClosed(DatabaseEvent event);
public void databaseItemAdded(DatabaseEvent event);
public void databaseItemDeleted(DatabaseEvent event);
public void databaseItemUpdated(DatabaseEvent event);	

< 표 2 자바빈즈 디자인 패턴을 적용한 메소드 >

생성한다. 프로퍼티를 생성하기 위해서 빈즈 클래스 코드를 생성해야 한다. 이 논문에서의 개인정보시스템에서 추출된 정보를 가지고 <표 2>와 같이 자바빈즈 디자인 패턴에 적용됨을 보인다.

3.4 레거시 코드와 자바 코드 매핑

자바네이티브 메소드로 생성된 래퍼는 <그림 5>와 같이 레거시 코드와 자바코드를 매핑하는데 사용하게 된다. 이 부분에서 객체 래핑 기법과 자바빈즈 디자인 패턴을 사용하여 레거시 코드를 자바빈즈 컴포넌트로 변환하게 된다. 여기서의 래퍼는 위에 자바네이티브 메소드로 변환한 과정을 포함하게 되고 공유 라이브러리를 통해 자바와 연계하게 된다.



< 그림 5 레거시 코드와 자바코드 연계 >

3.5 자바빈즈 컴포넌트 시각화

이 단계에서는 C++로 작성된 개인정보시스템의 주소록 일부를 자바와 연계하여 컴포넌트로 변환한 자바빈즈의 뷰 역할을 수행하는 사용자 인터페이스 부분에 관하여 나타낸다. 생성된 자바빈즈 컴포넌트는 JSP의 액션태그를 사용하여 기능을 제공하게 된다.

4. 성능평가

본 연구에서는 레거시 어플리케이션을 컴포넌트화하기 위해 제안된 자바빈즈 변환 프로세스 각 단계를 기준으로 기존의 방법들과 본 논문에서 제안하는 프로세스에 적용된 기법을 비교하여 <표 3>에 나타내었다.

	기존 방법	제안된 프로세스
어플리케이션 분석 단계	1. 분석과정이 해당 클래스 또는 해당 프로그램의 구성요소에만 국한되어 있음. 2. 클래스 내의 속성, 메소드, 이벤트에 한정된 분석을 함. 3. 분석, 설계 정보의 고려가 없음.	1. 기존 어플리케이션이 가지는 리소스뿐만 아니라 분석, 설계 정보를 바탕으로 분석 과정이 이루어짐. 2. 비즈니스 모델링과 시스템 모델링을 바탕으로 재사용성을 고려함. 3. 변환 과정이 이루어지기 전에 다이어그램을 통해 기존 메소드의 행위를 이해함으로써 컴포넌트 요소 정보 및 추가적인 정보를 파악함.
변환 단계	1. 문법에 의존한 클래스 구성 요소를 추출함. 2. 애플릿과 어플리케이션을 대상으로 함. 3. IDL 이용하여 래핑함으로써 분산 객체로 사용됨.	1. 재사용 단위의 추출은 어플리케이션 분석 단계에서의 이해를 통한 재사용하고자 하는 속성들에 한정됨. 2. 무조건적인 재사용을 피함으로써 변환 과정이 효율적이며, 오류가 적음. 3. 어플리케이션 변환만을 고려함. 4. WDL과 JNI를 사용하여 변환함으로써 다른언어와의 이식성을 제공함.
결과물	1. 재사용 지원을 위한 명세에 초점을 둠. 2. GUI의 미비한 재사용에만 초점을 둠	1. 변환 과정의 결과물, 변환 후의 결과물 중심의 연구에 초점을 둠. 2. 생성된 자바빈즈는 통합개발환경(IDE)을 통해 재사용이 용이함. 3. 비주얼하지 않은 컴포넌트도 고려함

< 표 3 기존 방법과의 비교 >

5. 결론

본 논문에서는 컴포넌트 기반의 소프트웨어를 개발하는데 있어서 효율성과 재사용성을 극대화시키기 위해 기존의 개발되어진 레거시 어플리케이션 자원을 이용하여 컴포넌트화시키는 분석 방법 및 변환 프로세스를 제안하였다. 이것은 재공학 방법을 사용하여 사용자 인터페이스 중심인지, 비즈니스 로직 분석중심으로 컴포넌트화할 것인지 따라 세분화시켜 명세하도록 하였으며, 다른 언어와의 연계 가능한 WDL과 자바네이티브 인터페이스를 사용하여 래퍼를 구현하였다.

그러나 본 연구에서 제안한 자바빈즈 변환 프로세스는 기존 어플리케이션을 대상으로 재사용이 이루어지므로, 대규모 시스템일수록 변환의 복잡도가 증가하고 코드 변환 단계에서의 프로그래밍 패턴을 포함하는 자바빈즈 디자인패턴의 추가와 자바네이티브 인터페이스로의 변환에 대한 코드의 양 증가와 수동적인 조작이 불가피하다.

향후 연구로는 제안된 각 단계의 체계적인 정립을 통한 세부적인 지침의 마련과 이의 검증이 필요하며, 변환을 위한 자동화 할 수 있는 툴에 대한 구체적인 연구가 이루어져야 할 것이다.

참고문헌

[1] Lawrence Wilkes, "Creating Components from Legacy Applications", CBDi Forum Journal, December 1998.

[2] Richard N. Taylor, et al. "A Component-and Message-Based Architectural Style for GUI Software", IEEE tansaction on Software Engineering, VOL 22, No. 6. June 1996.

[3] Gregory Abowd, Ashok Goel, Dean F, Jerding, Michae McCracken, Melody Moore, J. William Murdock, ColiPotts, Spencer Rugaber and Linda Wills. "MORALE-Missio Oriented Architectural Legacy Evolution." Poceddings International Conference on Software Maintenance'97, Bari, Italy, September 29-October 3, pp. 150-159, 1997.