컴포넌트 기반 개발을 위한 기존 애플리케이션 클래스의 JavaBean으로의 변환

김 병 준[†] · 김 지 영[††] · 김 행 곤[†††]

요 약

소프트웨어 부품화는 소프트웨어 개발에 있어서의 궁극적인 목표이다. 컴포넌트 기반의 개발은 이러한 재사용의 초점을 코드에 의한 수동적인 조작이나, 클래스 라이브러리보다 발전된 형태인 컴포넌트에 초점을 두고 있다. 컴포넌트 구축은 컴포넌트 모델에 적합한 새로운 소프트웨어 컴포넌트를 재개발 해야하는 비효율성으로 인해 추가적인 노력과 비용을 야기 시킨다. 최근 많이 사용되고 있는 자바 응용시스템의 경우 자바 언어 기반의 컴포넌트 모델이 존재하지만, 소규모의 재사용 단위나 제한된 GUI 컴포넌트 개발에만 머무르고 있어, 컴포넌트로의 기능을 충분히 발휘하지 못하거나 추가적인 비용, 노력이 필요하며 또한 특정 도메인 컴포넌트에서만 제한적이다는 단점이 있다. 따라서, 본 논문에서는 기존의 자바로 개발된 응용시스템을 기반 하여 자바의 컴포넌트 모델인 자바빈즈를 적용하기 위해, 컴포넌트를 확장 추출하고, 재사용단위로서 비즈니스 로직의 부분적인 수용을 통해 이 응용시스템에서의 자바빈즈로 변환하는 프로세스를 제시하고 알고리즘을 제안한다.

Transformation from Legacy Application Class to JavaBeans for Component Based Development

Byung Jun Kim[†] · Ji Young Kim[††] · Haeng kon Kim[†††]

ABSTRACT

Reusable software component is an ultimate goal for the software development. Component based development is focused on advanced concepts rather than passive manipulation or class library with source codes. However, the primary component construction in component based development lead to an additional development cost and effort for reconstructing the new software component within a component model. Java application provides several features based on component model. But, we only have an opportunity to develop the smallest reuse units or the restricted set of GUI components. It cannot contributed as a component and only used in the specific domain component with high cost and efforts. In this paper, we apply java component model to the existing java application and extract javabeans through extending the component scalability. We also discuss the algorithm for transformation mechanism from legacy class to javabeans with a partial of business logic.

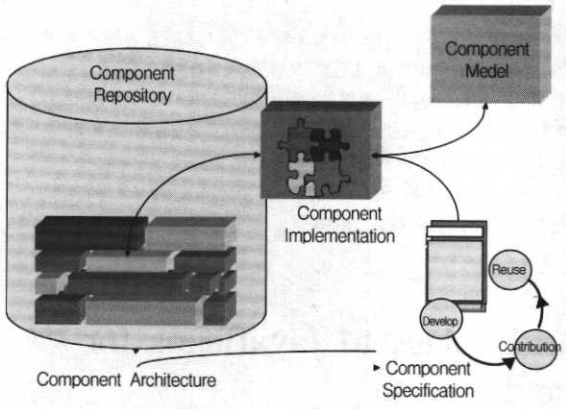
키워드 : 컴포넌트(Component), 자바빈즈(Javabeans), 재공학(Reengineering), 분류(Classfication), 설계서식(Design Pattern), 커스터마이징(Customizing)

1. 서 론

소프트웨어를 재사용 하고자 하는 소프트웨어공학의 목표는 코드 재사용, 객체, 라이브러리와 클래스에서 발전하여 컴포넌트에 초점을 두고 있다. 품질이 보증된 소프트웨어 부품을 규격화하고, 유통하는 기반 인프라를 마련하고, 이러한 소프트웨어 부품을 조합함으로써 주어진 시간이내에, 신뢰성을 가진 소프트웨어를 경제적으로 개발하는 것이 컴포넌트 기반 개발의 목표이다. 소프트웨어 부품, 즉 컴포넌트를 조합하여 애플리케이션을 생산하는 공학적 개념의 컴포넌트 기반 개발(CBD : Component Based Development)은 소프트웨어 개발의 적시성, 생산성 향상, 유지보수 및 확장성, 품질 향상 등 여러 가지 장점을 가지고 있다[1]. 이미 컴포넌트 기반 개발방법은 기존의 어플리케이션에서부터 웹 어플리케이션 개발에까지 반영되고 있으며, 국내에서도 KCSC (한국SW컴포넌트 컨소시엄)나 컴포넌트뱅크를 주축으로 컴포넌트 저장소 또는 유통, 판매사이트가 등장하고 있어 그 유용성을 입증 받고 있다. 컴포넌트 기반 개발은 (그림 1)에서와 같이 컴포넌트 아키텍처와 컴포넌트 모델, 그리고 이들 기반의 컴포넌트 명세와 명세에 따른 구현이 이루어져야 하며, 컴포넌트의 관리와 배포, 조립에 의한 응용 생성을 지원하는 컴포넌트 저장소등의 기반구조가 필수적이다. 그러나, 컴포넌트 기반 개발에 있어 중요한 한 요소인 컴포넌

† 준 회 원 : (주)라스21 연구개발부
†† 준 회 원 : 대구가톨릭대학교 대학원 전산통계학과
††† 종신회원 : 대구가톨릭대학교 컴퓨터정보통신공학부 교수
논문접수 : 2001년 12월 19일, 심사완료 : 2002년 5월 14일

트 구축은 소프트웨어의 재사용을 위해 컴포넌트 모델에 따른 새로운 소프트웨어 부품을 다시 만들어야 하는 비효율성을 가진다. 이러한 이유로 신뢰성을 보장받은 기존의 어플리케이션을 대상으로 컴포넌트를 추출하거나 컴포넌트화하는 행위는 비교적 빠른 구축을 이룰 수 있으며, 검증된 비즈니스 로직과 추가적으로 발생할 수 있는 장점을 가질 수 있다[2, 3].



(그림 1) CBD Process의 구성요소

일반적으로, 컴포넌트는 여러 가지 상황과 환경에서 일관된 동작을 수행하며, 미리 정의된 인터페이스를 통해 컨테이너에 배치되며 다른 컴포넌트들과 상호 운영되어, 사용자의 빌더에 적용됨으로써 원할한 재사용을 지원한다. 따라서 재사용의 효율성은 그 재사용 단위인 컴포넌트 모델의 정규화된 실용적 접근에 의존한다고 볼 수 있다. 이러한 관점에서 순수 객체지향언어이자 플랫폼 독립적인 자바언어의 컴포넌트 모델인 자바빈즈는 아주 적합한 컴포넌트 모델이다. 자바빈즈는 단일하고 고성능의 API를 제공하며, 단순하여 사용하기 쉽다. 하지만 기존의 자바빈즈 컴포넌트의 주 활용범위는 RAD(Rapid Application Development) 도구나 IDE (Integrated Development Environment)에서 사용자 인터페이스 정도의 미비한 재사용에 불과했다. 그러나 이렇게 소규모 컴포넌트 단위에 한정된 재사용은 자바로 작성된 기존의 어플리케이션으로부터 비즈니스 로직의 패키징이나 도메인 로직의 부분적인 수용으로 재사용의 폭을 넓힐 수 있다[4].

본 논문에서는 컴포넌트 기반 개발에서의 컴포넌트 구축을 지원하기 위해 자바로 작성된 기존의 어플리케이션을 자바빈즈로 변환하는 기법을 제시하고자 한다. 즉, 자바로 작성된 기존의 어플리케이션을 컴포넌트의 행위적인 측면에서 분석하여, 자바빈즈의 설계 서식에 따라 컴포넌트화의 범위를 최대화하고, 재사용단위로서 도메인 로직의 부분적인 수용을 통해 중간 규모의 컴포넌트 구축을 용이하게 하고자 한다. 따라서, 궁극적으로 컴포넌트 기반 개발의 지원을 위한 컴포넌트 구축에서의 효과적인 재사용을 이루고자 한다. 2장에서는 자바빈즈에 대한 관련연구와 컴포넌트 추출 및 변환에 대해 기술하고, 3장에서는 자바빈즈 변환 프로세스와 각 단계별 과정, 결과물을 제시하며, 4장에서는 기존 자바 어플리케이션으로부터 자바빈즈 컴포넌트 구현과 5장의 결론 및 향후 연구로 정리하였다.

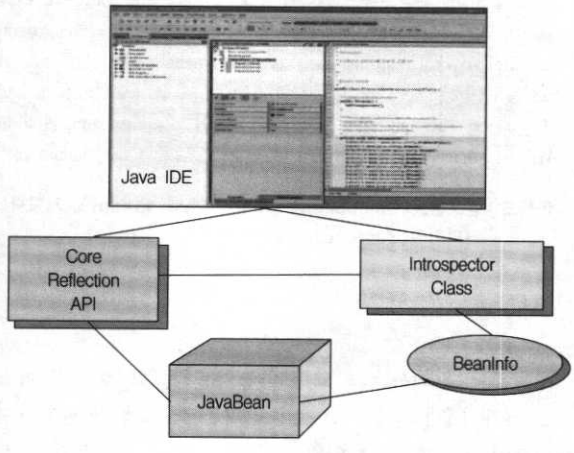
2. 관련 연구

2.1 자바빈즈

자바빈즈는 자바 언어를 위한 소프트웨어 컴포넌트 모델로써 빌더 도구에서 시각적으로 조작할 수 있는 재사용 가능한 소프트웨어 컴포넌트이다. 또한 컴포넌트 개발자가 생산, 유통하여 최종 사용자가 재사용할 수 있도록 고안되었으며 Sun사에서 제안한 플랫폼 중립적인 소프트웨어 컴포넌트 아키텍처를 기반으로 한다. 자바빈즈와 함께 현재 대중적으로 사용되어지는 컴포넌트 모델과 서비스 레벨은 <표 1>과 같다. (그림 2)는 자바개발도구에서 자바빈즈를 사용하기 위해 시각화하는 것을 도식화한 것이다.

〈표 1〉 서비스 레벨에 따른 컴포넌트 모델

	COM	Java	CORBA
Basic components	Com components	JavaBeans	CORBA objects
Protocol	DCOM	RMI	CORBA IIOP
Enterprise srvices	COM+	EJB/J2EE	CORBAServices



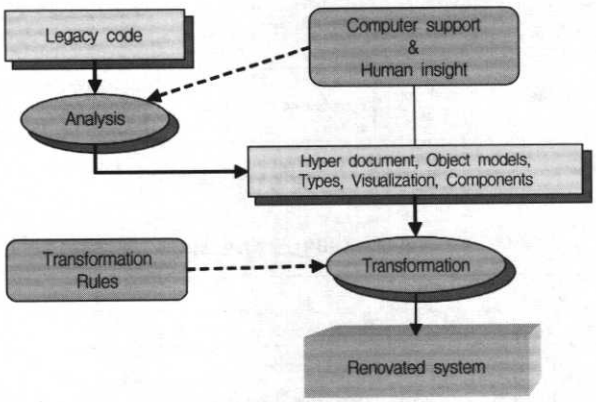
(그림 2) 개발도구에서의 관점

자바빈즈는 <표 2>에 명시된 명명규칙에 따라 작성됨으로써, 코어 리플렉션(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. 또한, 자바 빈즈의 구현에는 직렬화(Serialization), 명명 관례, 설계 관례 등 몇 가지 세부사항을 지침

해 두고 있다[5, 6].

〈표 2〉 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)



(그림 3) 레거시 시스템의 분석과 변환

자바빈즈는 최종사용자의 IDE에 배치되어 요구되는 동작 수행을 목표로 하므로, 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[7-9].

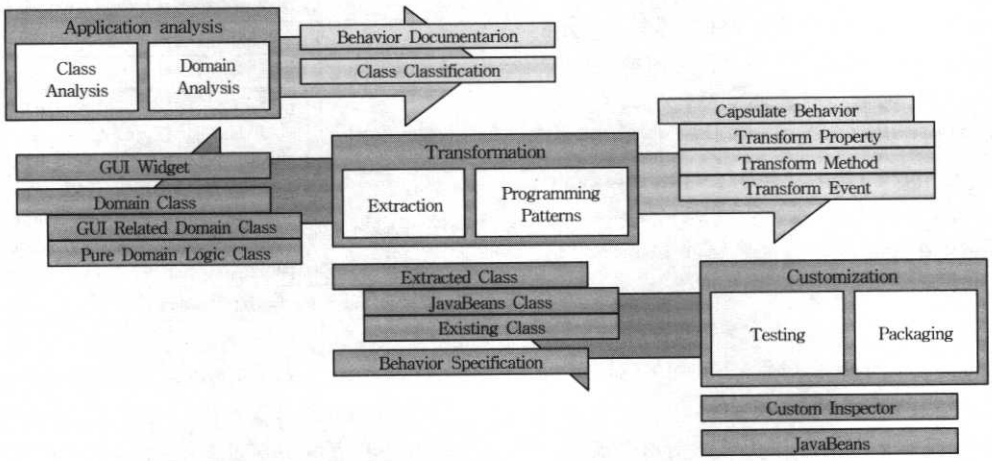
2.2 컴포넌트 추출 및 변환

CBD는 고품질의 컴포넌트 구축과 컴포넌트 아키텍처 또는 컴포넌트 프레임웍의 용이한 구성으로 완성된다. 고품질의 컴포넌트 구축에는 일반적으로 어플리케이션 개발과 상이한 목적으로 특정 컴포넌트 모델과 플랫폼을 선정, 새로운 컴포넌트를 구축하는 관점과 이미 기업환경이나 최종사용자들에게 그 품질을 인정받은 기존 어플리케이션으로부터 가치있는 비즈니스 로직을 추출하여 컴포넌트화 하는 관점, 두 가지를 들 수 있다. 기존의 어플리케이션으로부터 컴포넌트를 추출하거나 재구축하는 경우 상대적으로 프로그램 품질의 신뢰도 향상과 개발 사이클의 감소, 향후 재사용에 따른 부가가치 등 여러 가지 이점을 가질 수 있다[10, 11]. (그림 3)은 레거시 시스템에 새로운 비즈니스 요구를 반영하기 위한 변환과정을 나타낸다. 기존 코드를 분석하는 과정은 자동화된 코드분석기나 전문가의 관점에서 수행되며, 이를 통해 Hyper 문서, 오브젝트 모델, 타입, 시각적 요소, 컴포넌트들이 추출되며, 목적 플랫폼의 변환규칙에 따라 변환이 이루어짐으로써 새로운 시스템으로 재개발된다[12, 13].

3. 자바빈즈로의 변환

3.1 개 요

기존 어플리케이션으로부터 추출되어 작성된 컴포넌트가 배치될 환경에서 무리 없이 요구되는 행위와 도메인 로직을 수행하기 위해서 가장 중요한 것은 구축된 컴포넌트 모델과 배치된 컴포넌트 아키텍처와의 관계이다. 이러한 관점



(그림 4) 자바빈즈 변환 프로세스

에서 자바로 작성된 어플리케이션은 자바언어의 컴포넌트 모델인 자바빈즈로 변환하여 재사용 하는데 있어 무리가 없다. 본 논문에서 제안하는 컴포넌트 변환 프로세스는 자바 기반의 어플리케이션으로부터 자바의 컴포넌트 모델인 자바빈즈로 추출, 구축함으로써 컴포넌트를 설계, 구현, 사용함에 있어 잠재적인 오류를 줄여준다.

3.2 JavaBeans 변환 프로세스

본 논문에서 제안하는 자바빈즈 변환 프로세스(그림 4)는 자바로 작성된 기존의 어플리케이션을 설계단계의 결과물들과 구현물인 코드를 바탕으로 한 분석과정을 수행한다. 클래스 다이어그램을 통해 어플리케이션의 각 클래스들의 상호작용을 파악하고, 적절한 분해과정을 거친 후, 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 프로세스는 기존에 자바로 작성되어진 어플리케이션을 전제로 해서 자바빈즈로의 변환을 목적으로 한다.

〈표 3〉 클래스 형태에 따른 분류

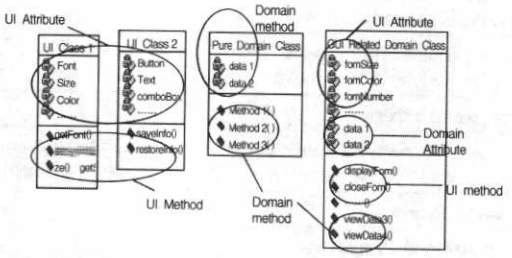
구 분	형 태
UI Class (분류1)	UI를 구성하는 속성들과 메소드로 이루어진 클래스, 자바빈즈로 변환 가능하거나 자바빈즈의 형태로 담겨진 클래스
Pure Domain Class (분류2)	순수하게 도메인 로직을 수행하는 클래스, UI Class와 상호작용이 있을수 있으며, Invisibel 빈의 성격을 가짐
GUI Related Domain Class (분류3)	UI를 구성하는 속성과 메소드에 도메인 로직을 포함하는 클래스

Step 1 : 어플리케이션 분석은 변환하고자 하는 어플리케이션의 목적 도메인의 관점에서 클래스를 분류하여 UI widget이나 순수 도메인 로직 클래스, UI와 관련되어 도메인 로직을 수행하는 클래스를 추출해내며 어플리케이션의 행위적인 관점에서의 식별과 문서화 과정을 가진다. 어플리케이션 분석은 두 가지 행위, 즉 클래스 분석과 기존 어플리케이션의 도메인 분석으로 나뉜다.

클래스 분석은 어플리케이션을 구성하는 여러 클래스들간의 상호관계를 파악하고, 이들중 재사용 가능한 부분을 추출해 내기 위한 과정으로 클래스의 형태에 따라 위의 <표 3>과 같이 분류되어 진다.

일반적으로 UI를 구성하는 i) 클래스(분류 1)는 그 자체로 빈즈가 될 수 있으나 그 규모가 너무 작고 이미 대부분의 IDE에서 사용이 일반화되어 있는 추세이다.

두 번째 ii) 순수 도메인 로직을 수행하는 클래스(분류 2)는 Invisible 빈의 성격을 띠며 (분류 1)의 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 (분류 3)의 클래스들에 초점을 두고 있다.



클 래 스	메 소 드	분 류	연관 클래스	행 위
UI Class 1	setFont( ) setColor( )	Classification 1	UI Class 2 PD Class	GUI Representation
UI Class 2	saveInfo( ) restoreInfo( )	Classification 1	PD Class	GUI Representation Save/Restore
Pure domain Class	Method 1( ) Method 2( ) Method 3( )	Classification 2	UI Class 1 UI Class 2 UI Related Domain Class	Domain Logic Save/restore
UI related Domain Class	displayForm( ) viewData 1( )	Classification 3	PD Class	GUI Representation Domain Logic Save/Restore

(그림 5) 행위적 관점에서의 클래스 분류

〈표 4〉 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 PropertyVetoableException ; 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	

iii) 도메인 로직을 포함하는 UI 클래스(분류 3)는 UI를 구성하는 클래스에 부분적인 도메인 로직을 담고 있는 형태로써 적절한 클래스 분리를 통해 보편적인 컴포넌트 형태로 구축될 수 있다.

(그림 5)는 어플리케이션 분석단계에서 기존 어플리케이션의 클래스 다이어그램을 참조로 클래스들을 행위적 관점에서 분류하고, 연관 클래스와 클래스의 행위를 분석해놓은 행위 분석표(Behavior Documentation)이다. 또한 도메인 분석을 통해 선택된 어플리케이션에서 추출하고자 하는 컴포넌트의 기능이 일반적인 도메인 관점에서 재사용이 타당한 것인가를 이 단계에서 판단하고 고려한다.

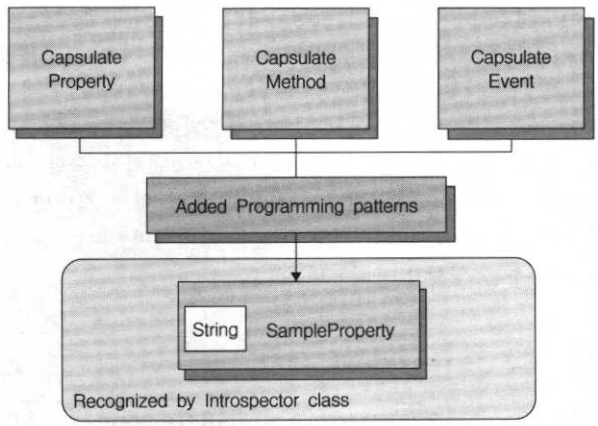
Step 2 : 변환은 본 프로세스의 코어로써 어플리케이션 분석단계의 결과인 클래스 분류(Class Classification)와 행위 분석표(Behavior Documentation)를 바탕으로 빈즈로의 변환 과정을 가진다.

추출될 클래스를 앞 단계의 결과물들을 바탕으로 선정하고, 자바빈즈의 프로그래밍 패턴에 따라 추출 클래스들의 변환을 위해 속성, 메소드, 이벤트의 행위적인 캡슐화가 이루어지며 필요에 따라 추가되거나 삭제되는 요소들이 있을 수 있다. <표 4>는 자바빈즈 변환에 있어서 기본적으로 추가되어야 할 서식을 나타내며 이에 입각한 코드 매핑이 이루어진다. 이러한 설계서식은 클래스와 메소드 명칭, 클래스 상속, 인터페이스의 구현, 메소드 인자 유형, 그리고 부분적 혹은 전체적으로 클래스나 메소드의 사용을 인지하기 위해 사용될 수 있는 예외들의 특징이다. 설계 서식은 표준 속성 패턴을 Simple, 논리 속성 패턴을 Boolean으로 정의한다. Indexed는 클래스간의 상호작용에 사용되며, 하나의 속성 변화시 다른 클래스나 속성에게 통지할 목적으로 Bound와 Constrained 속성의 지원이 필요하다. 이벤트 전달에 있어서는 일대일 이벤트 전달을 unicast로, 일대다를 multicast 형태로 제공된다.

다음의 <표 5>는 변환단계에서 이루어지는 속성과 메소드에 대한 기본적인 코드 매핑을 나타내고 있다. String으로 선언된 SampleProperty를 자바빈즈의 속성으로 사용하기 위해 <표 4>의 설계 서식에 따라 추가한 코드 매핑의 예이다(그림 6).

〈표 5〉 코드 매핑 예

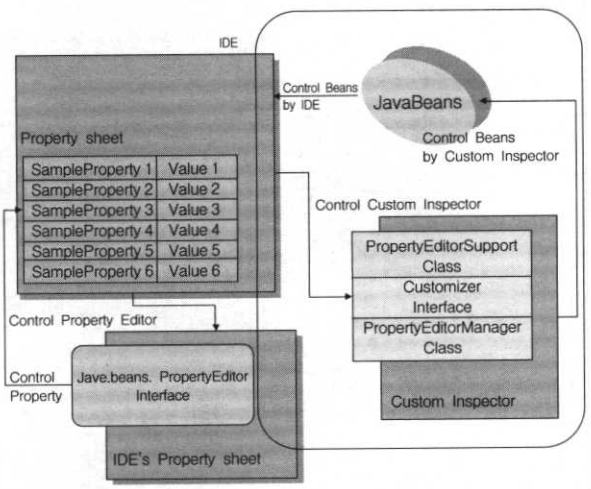
```
private String SampleProperty ;
private static final String SAMPLE_PROPERTY =
        "SampleProperty" ;
private String sampleProperty ;
private PropertyChangeSupport propertySupport ;
public testBean() {
propertySupport = new PropertyChangeSupport ( this ) ; }
public String getSampleProperty () {
return sampleProperty ; }
public void setSampleProperty (String value) {
String oldValue = sampleProperty ;
sampleProperty = value ;
propertySupport.firePropertyChange
        (SAMPLE_PROPERTY, oldValue, sampleProperty) ; }
public void addPropertyChangeListener
        (PropertyChangeListener listener) {
propertySupport.addPropertyChangeListener (listener) ; }
public void removePropertyChangeListener
        (PropertyChangeListener listener) {
propertySupport.removePropertyChangeListener (listener) ; }
}
```



(그림 6) 속성의 캡슐화

Step 3 : 커스터마이징은 변환된 자바빈즈를 그 목적에 맞게 IDE나 최종사용자가 사용할 수 있도록 지원하는 단계이다. 변환된 자바빈즈의 특성에 따라 커스텀 인스펙터(Custom Inspector)가 작성되어 질 수 있으며, 테스팅, 배포를 위한 패키징 과정과 자바빈즈에 대한 문서화를 포함한다.

커스텀 인스펙터는 IDE와 자바빈즈를 지원하는 응용프로그램에서 사용자를 지원하기 위해 빈즈의 기본 속성은 물론, 확장된 속성의 상태창을 제공한다. 다음의 (그림 7)은 IDE에서의 자바빈즈의 기본 속성을 제어하는 일반적인 속성 편집기와 본 논문에서 제안하는 커스텀 인스펙터에서의 빈 제어를 도식화한 것이다. 기존의 어플리케이션에서 컴포넌트를 추출해내는 경우, 어플리케이션의 도메인에 따라 기본형 속성보다는 확장된 클래스 속성의 유형이 일반적으로 사용되어지기 때문에 IDE의 일관적인 속성 편집기로는 빈즈의 제어가 충분치 못하다. 따라서 변환된 자바빈즈의 효과적인 재사용을 지원하기 위해서 java.beans 클래스의 Customizer 인터페이스를 구현하여 자바빈즈 전체를 제어할 수 있는 커스텀 인스펙터를 작성해주어야 한다.



(그림 7) 커스텀 인스펙터에 의한 빈의 제어

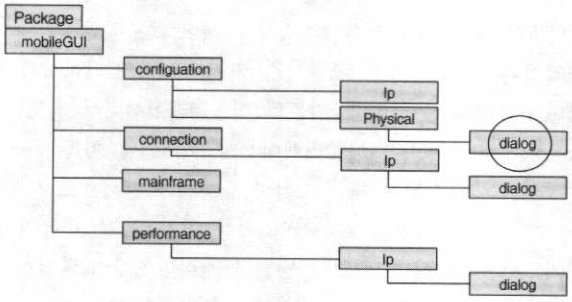
커스터마이징 단계에서 수행되어야 할 마지막 과정은 변환된 자바빈즈를 사용자나 IDE에 배포하고, 적절히 사용할 수 있도록 지원하는 패키징 과정이다. 표준적으로, 자바빈즈를 배포하고, IDE에 배치되는데 사용되어지는 방법은 Jar 패키징이다. Jar은 자바빈즈를 배포하는 기본적인 수단이며, 하나의 자바빈즈 혹은 빈과 밀접한 연관이 있는 자바빈 집합, 클래스 파일 그리고 지원 파일을 포함하며 개발자를 대상으로 하는 명세가 포함된다.

4. 사례 연구

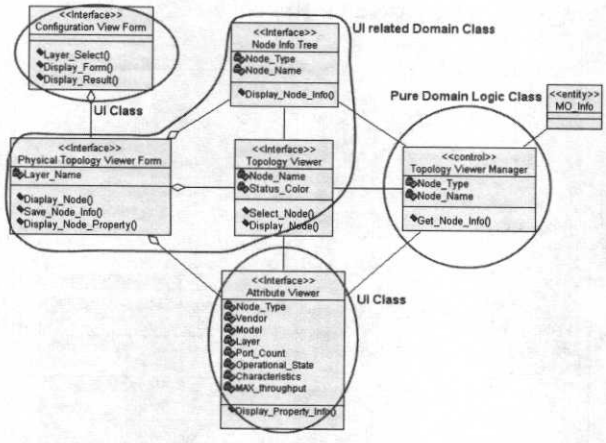
4.1 망관리 사용자 인터페이스의 자바빈즈 변환

본 논문에서는 기존에 자바로 작성된 NMS시스템에서 망관리를 수행하는 어플리케이션을 제안된 변환 프로세스를 적용하여 자바빈즈로의 변환을 구현하였다.

기존 어플리케이션에 대한 정보로써 (그림 8)의 패키지 상관도와 (그림 9)의 망관리 영역 중 구성관리 사용자 인터페이스의 노드 정보보기의 클래스 다이어그램을 참고하였다[14-16]. 이를 통해 개략적으로 UI 클래스, 도메인 로직과 UI표현을 동시에 수행하는 클래스, 그리고 순수하게 도메인 로직만을 수행하는 Control 클래스로 분리할 수 있다.



(그림 8) 망관리 사용자 인터페이스의 패키지 상관도



(그림 9) 클래스 분리

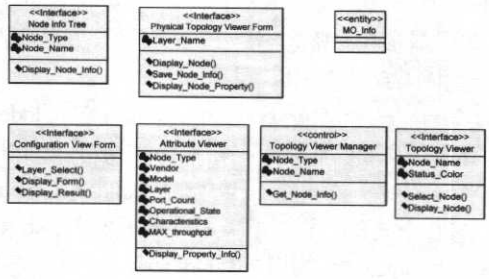
4.2 어플리케이션 분석

기존 어플리케이션의 분석 정보인 패키지 상관도와 클래스 다이어그램을 토대로 어플리케이션의 클래스들을 분리할 수 있다. 이를 바탕으로 어플리케이션 분석단계의 클래스 분류표를 도출해낼 수 있다<표 6>.

〈표 6〉 클래스 분류

구 분	클 래 스	연관 클래스
UI Class (분류 1)	PhysicalLayerPanel DrawNode, DrawLink	DrawGridScrollPane
Pure Domain Class (분류 2)	DrawObject DrawObjectLinkedList Module	DrawNode, DrawLink
GUI Related Domain Class (분류 3)	DrawGridScrollPane SetNodeAttributeDialog SetLinkAttributeDialog	DrawGridScrollPane

클래스 분류의 결과를 고려하여, 패키지내 전체 클래스를 클래스 자체의 행위적인 관점에서 분류하고, 연관 클래스와 메소드를 분석함으로써 (그림 10)의 행위분석표를 작성하였다.



no	클래스	메소드	분류	연관 클래스	행위
1	NodeInfoTree	boolean isNamingContext(NameComponent p[ ]) public String[ ] getChildName(String[ ] strPathAndNodeName) public void createbasetree()	Classification 3	2, 4	GUI Representation Domain Login
2	Physical Topology ViewerForm	void Button_actionPerformed(ActionEvent e)	Classification 1	1, 4, 5	GUI Representation
3	AttrivuteViewer	Public void showAttribute(int x, int 4, mo currentMO) Public void actionPerfarmed(ActionEvent e)	Classification 3	5	GUI Representation Domain Login
4	TopologyViewerM	private void initValue() private void calcConnection()	Classification 2	1, 2, 5	Domain Login Save/Restore
5	TopologyViewer	private Point moveConnection (int x, int y) private void selectObject (int x, int y) private void paintComponent (Graphics g)	Classification 3	2, 3, 4	GUI Representation Domain Login
...	...	...	...	...	...

(그림 10) 행위 분석표

4.3 변 환

분류된 각각의 클래스들 중에서 클래스간의 연관성, 상호작용의 복잡도와 재사용성을 고려하여, 구성관리 패키지의 attributeViewer 클래스를 변환에 적용할 클래스로 추출하였다. 변환단계의 코드 변환은 <표 2>의 디자인 패턴과 <표 4>의 설계 서식을 적용하여 자바빈즈 코드로 매핑한다. 위의 <표 7>은 노드 속성에 대한 부가적인 정보의 서술을 위한 Description 속성에 대한 코드 매핑 테이블이다. 자바빈즈 변환을 위해 기본적으로 Serializable을 구현하였고, IDE에서 descriptionTextArea를 인식할 수 있도록 속성의

캡슐화를 수행하였다. 캡슐화된 속성에 대한 IDE의 조작을 위해 속성 리스너 인터페이스의 구현과 set/get 서식을 적용하였다[17-21].

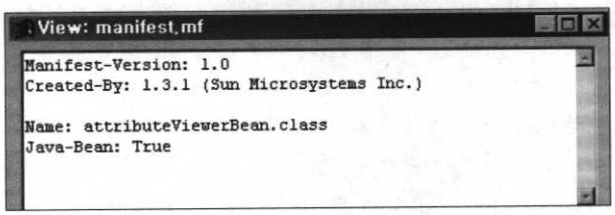
〈표 7〉 코드 변환

Legacy code	생성자	public SetNodeAttributeDialog (JFrame parent, DrawObject n)
	속 성	private JTextArea descriptionTextArea ;
	선 언	descriptionTextArea = new JTextArea() ;
	속성 접근 서 식	descriptionTextArea.setText (objCurrentNode.m_description) ;
Transformed code	생성자	public SetNodeAttributeDialog()
	Serializable 구 현	public class attributeViewerBean extends JPanel implements java.io.Serializable, ActionListener {
	속성의 캡슐화	private static final String PROP_SAMPLE_PROPERTY = "SampleProperty" ; private String sampleProperty ; private PropertyChangeSupport propertySupport ; private JTextArea descriptionTextArea ;
	선 언	descriptionTextArea = new JTextArea() ; propertySupportSupport = new PropertyChangeSupport(this) ;

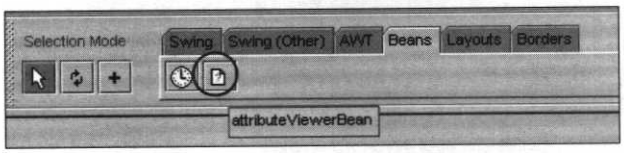
	속성 접근 서 식	public String getSampleProperty () {
		return sampleProperty ; }
		public void setSampleProperty (String value) {
		String oldValue = sampleProperty ;
		sampleProperty = value ;
		descriptionTextArea.setText(sampleProperty) ;
		propertySupport.firePropertyChange (PROP_SAMPLE_PROPERTY, oldValue, sampleProperty) ; }

4.4 테스팅과 커스터마이징

커스터마이징 단계에서는 변환된 노드 속성 정보 클래스와 jar파일내에 포함되는 파일들에 대한 정보를 가지는 manifest (그림 11)파일을 패키징한다[17].

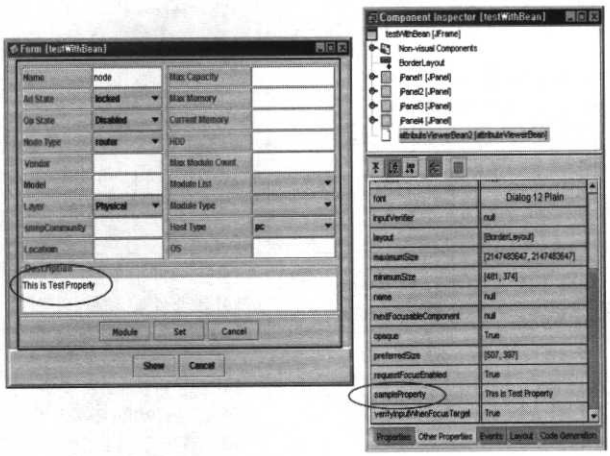


(그림 11) attributeViewer의 manifest file



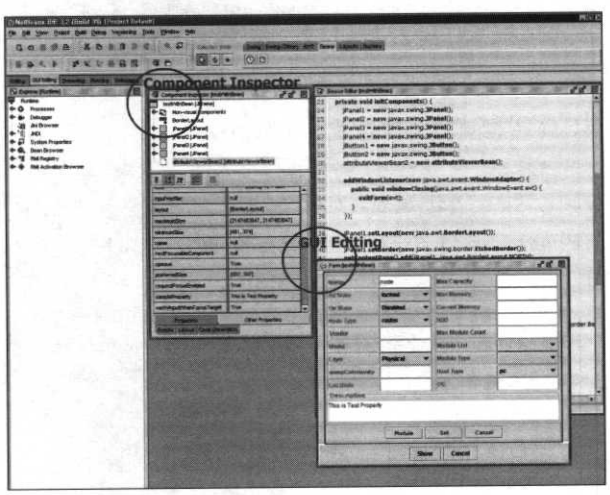
(그림 12) 컴포넌트 파레트

위의 (그림 12)은 jar로 패키징된 attributeViewer 빈즈를 자바 IDE의 컴포넌트 파레트에 인스톨한 것이다. 이렇게 설치된 자바빈즈는 IDE에 의해 인식되고, 기본적으로 제공되는 컴포넌트 인스펙터를 통해 속성과 메소드의 제어가 가능하다(그림 13).



(그림 13) 변환된 자바빈즈와 컴포넌트 인스펙터

아래 (그림 14)는 인스톨된 자바빈즈를 새로운 프레임에 배치하여 새로운 소프트웨어를 개발하는 IDE의 전체 외형을 나타낸다. 변환에 사용되어진 구현환경으로는 JDK1.2.2로 작성되어진 기존 어플리케이션의 노드 속성창을 JDK1.3.1으로 upgrade한 후 변환 프로세스의 각 단계를 수행하고, 설계서식을 적용하였으며, 구현은 JDK 1.3.1에서, 적용된 IDE는 OpenSource 프로젝트 IDE인 NetBeans 3.2를 사용하였다.



(그림 14) 변환된 빈즈의 재사용

4.5 변환 프로세스 평가

본 논문에서 제안된 자바빈즈 변환 프로세스는 기존의 자바 클래스를 컴포넌트 형태로의 재사용을 목적으로 한다. 다음의 <표 8>은 프로세스 각 단계를 기준으로 기존의 방

법들과 비교한다.

〈표 8〉 기존 방법과의 비교

	기존 방법	제안된 프로세스
어플리케이션 분석 단계	• 분석과정이 해당 클래스 또는 해당 프로그램의 구성요소에만 국한되어 있음 • 클래스내의 속성, 메소드, 이벤트에 한정된분석. • 분석, 설계 정보의 고려 없음	• 기존 어플리케이션이 가지는 바이너리 형태의 리소스 뿐만 아니라 분석, 설계 정보를 바탕으로 분석과정이 이루어짐. • 도메인 로직과 클래스사이의 행위를 바탕으로 재사용성 고려. • 변환과정이 이루어지기 전에 클래스 다이어그램을 통해 기존 클래스의 행위 이해
변환 단계	• 문법에 의존한 클래스 구성 요소의 추출, • 애플릿과 어플리케이션을 대상으로 함.	• 재사용 단위의 추출은 어플리케이션 분석 단계에서의 이해를 통한 재 사용하고자 하는 속성들에 한정됨. 무조건적인 재사용을 피함으로써 변환 과정이 효율적이며, 오류가 적음, • 어플리케이션 변환만을 고려.
결과물	• 재사용 지원을 위한 명세에 초점	• 변환과정의 결과물과 변환 후의 결과물 중심의 연구, • 변환된 자바빈즈는 IDE를 통해 재사용 용이

5. 코드 매퍼 프로토타이핑

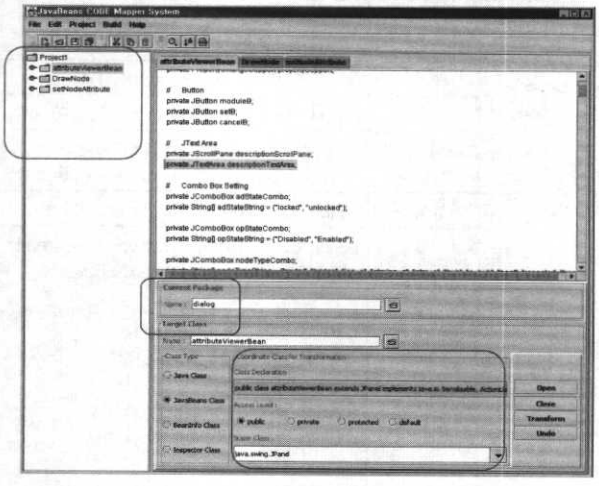
본 논문에서 제안하는 자바빈즈 변환 프로세스의 변환 단계는 자바빈즈 컴포넌트 모델로의 변환을 위해 기존 소스 코드에 물리적인 변경이 필수적이다. 코드 변환에는 재사용을 위해 속성, 메소드, 이벤트의 구분과 설계 서식의 추가들이 필수적으로 필요한데, 이러한 과정에서 코드의 추가가 필수적이며 재사용 단위가 증가할수록 캡슐화를 위한 코드의 양도 부가적으로 증가하게 된다.

따라서, 본 논문에서는 캡슐화를 위한 템플릿 코드를 제공하고 코드 변환 과정에서의 코딩의 번거러움을 해소하며 기본적인 자바빈즈 설계서식 제공을 목표로 하는 코드 매퍼(그림 15)를 프로토타이핑 해보았다.

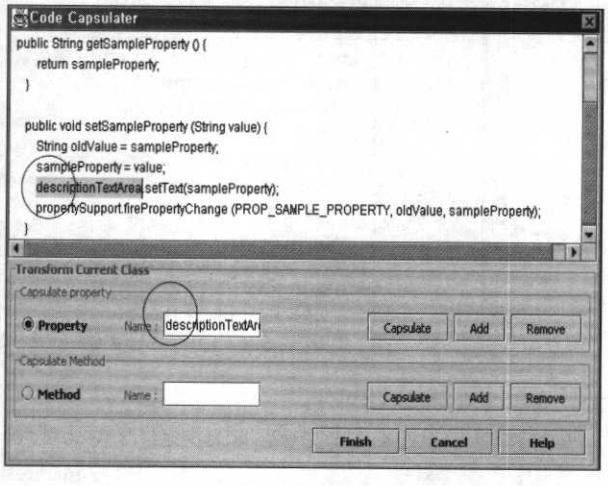
코드 매퍼는 다음과 같은 화면 구성으로 나뉘어진다.

- 프로젝트창
- 패키지 선택창
- 코디네이터 클래스 창
- 코드 에디터

프로젝트창에서는 현재 프로젝트내의 클래스와 각각의 클래스 메소드, 속성을 나열해서 보여주고, 패키지 선택창에서는 해당 패키지와 타겟 클래스를 선택한다. 이렇게 선택된 클래스는 코디네이터 창을통해 세부 정보를 선택하거나 수정하게 된다. 코드 캡슐레이터(그림 16)에서는 재사용하고자 하는 속성을 추가하거나 삭제하고 캡슐레이션에 대한 템플릿 코드가 제공된다. 마지막으로, 제공된 템플릿 코드는 재사용 속성에 맞게 편집되어 코드 매퍼의 에디터에 추가되게 된다.



(그림 15) 코드 매퍼



(그림 16) 캡슐화를 위한 코드 캡슐레이터

코드 매퍼는 변환단계의 코드 변환과정에서 불가피하게 발생하는 자바빈즈 설계서식의 추가나 프로그래밍 패턴 적용의 번거러움을 피하고, 변환의 자동화를 지원하기 위해 제안된다. 코드 매퍼는 프로그래밍 패턴을 포함하는 자바빈즈 설계서식을 제공하고, 재사용 단위들의 캡슐화를 위한 템플릿 코드를 제공한다. 따라서 기존 자바 클래스의 자바빈즈 변환에서의 자동화 추구에 보다 효율성을 제공하고자 한다.

6. 결 론

소프트웨어 환경과 요구 변화에 따라 발전된 수많은 개발 패러다임중에서도 현재의 컴포넌트 기반의 소프트웨어

개발은 소프트웨어 개발과 운영, 유지보수에 있어 최상의 경쟁적인 방법으로 인식되고 빠르게 적용되어지고 있다. 이미 컴포넌트 유통을 목적으로 하는 컴포넌트 유통 사이트의 출현과 e-Business에서의 CBD 도입은 이를 증명하는 예이기도 하다. 또한, 컴포넌트 저장소가 속속들이 구축되어지고 있는 현 시점에서 기존의 자바 어플리케이션으로부터 자바빈즈를 구축해나가는 방법은 보다 경쟁적이라 할 수 있겠다.

본 논문에서는 CBD 지원을 위해 응용의 빠른 개발과 생산성을 확보하고, 컴포넌트 구축에 있어서 보다 경쟁적인 방법으로써 기존 어플리케이션을 컴포넌트화하는 기법을 제안하고자 하였다. 따라서, 자바로 작성된 기존 어플리케이션을 자바언어의 컴포넌트 모델이자 범용적인 컴포넌트 모델인 자바빈즈를 채택하여 자바빈즈 컴포넌트로 변환하는 자바빈즈 변환 프로세스를 제안한다.

자바빈즈 변환 프로세스는 기존 어플리케이션이 가지는 분석, 설계물을 비롯한 모든 종류의 리소스를 바탕으로 어플리케이션 분석, 변환, 커스터마이징 단계를 통해 기존의 클래스를 자바빈즈 컴포넌트로 변환한다.

본 논문에서는 제안된 자바빈즈 변환 프로세스의 사례연구로써 자바로 개발된 기존의 NMS 시스템의 망관리 사용자 인터페이스에 자바빈즈 변환 프로세스를 적용 자바빈즈를 구현해 보았다. 그 결과로 클래스나 소스라인 에디팅을 통한 재사용이 아닌 컴포넌트 형태로의 재사용이 용이함을 확인하였으며, 다음과 같은 장점을 가진다.

- 동일 수행환경(자바 가상 머신)에 의한 잠재적인 오류 감소
- 컴포넌트 형태의 재사용을 통한 명확하고, 자동화된 소프트웨어 재사용 지원

이러한 컴포넌트 형태로의 재사용은 어플리케이션 개발자들의 직관적인 API 인식과 IDE에서의 디자인과 런타임시의 결과물의 일치화를 제공하므로 CBD 본래의 목적에 적합하게 부합한다고 볼 수 있다. 또한 변환단계의 코드 변환과정에서 불가피하게 발생하는 자바빈즈 설계서식의 추가나 프로그래밍 패턴의 적용을 피하고, 변환의 자동화를 지원하기 위해 코드 매퍼를 프로토타이핑하였다. 코드 매퍼는 프로그래밍 패턴을 포함하는 자바빈즈 설계서식을 제공하고, 재사용 단위들의 캡슐화를 위한 템플릿 코드를 제공한다.

자바빈즈 변환 프로세스의 단점으로는 기존 어플리케이션을 대상으로 재사용이 이루어지므로, 대규모 어플리케이션일수록 변환의 복잡도가 증가하고, 클래스 자체의 재사용성이 떨어지는 현상이 발생한다. 또한, 코드 변환 단계에서 프로그래밍 패턴을 포함하는 자바빈즈 설계서식의 추가는 코드의 양 증가와 수동적인 조작이 불가피하다.

향후 연구로는 제안된 변환 프로세스 각 단계의 체계적인 정립을 통한 세부적인 지침의 마련과 이의 검증이 필요하며, 변환을 위한 CASE Tool에 대한 구체적인 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] Wilkes, Lawrence, Understanding Component Based Development, Addison-Wesley, June, 2000.

[2] Luqi, Jiang Guo, "Toward Automated Retrieve for a Software Component Repository," IEEE Conference and Workshop on Engineering of Computer-based Systems, March, 1999.

[3] 김행곤 외, 컴포넌트 저장소 형상 관리 시스템 개발, 중간연구보고서, 한국전자통신연구원, 2000.

[4] Morgan Kinne, "Writing JavaBean Property Editors," Dr. Dobb's, Vol.24, No.2, Feb., 1999.

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

[6] Mary Campione 외, The Java Tutorial, SUN, 2000.

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

[8] Reaz Hoque, Programming JavaBeans 1.1, McGraw- Hill, 1998.

[9] "Implementation of a DEVS-JavaBean Simulation Environment," yung-Hsin Wang, Proceedings of the 34th Annual Simulation Symposium, pp.333-338, April, 2001.

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

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

[12] Arie van Deursen, "From Legacy to Component : Software Renovation in Three Steps," CWI, 2000.

[13] Michael Richmond, "Component Migration with Enterprise JavaBeans," ACM, pp.79-80, Oct., 2000.

[14] 김행곤외, 차세대 인터넷/인트라넷 네트워킹 시스템 개발 연구, 기술보고서, 2001.

[15] 김행곤외, "망관리 사용자 인터페이스 시스템 구축을 위한 컴포넌트 명세 및 프로토타이핑", 정보처리학회 제5회 산학연 소프트웨어공학기술대회, 2001.

[16] Fowler-Scott, UML Distilled, Addison-Wesley, 1997.

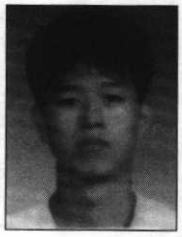
[17] Sun Microsystems, Inc, "How to be a Good Bean," 1997.

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

[19] Kathy Wakrath외, The JFC Swing Tutorial, SUN&정보문화사, 2000.

[20] Gray Cornell 외, core JAVA, SUN & 영한출판사, 1998.

[21] Beth Stearns, JavaBeansTM 101 Part I, II : Writing a Simple JavaBean, 2000. http://developer.java.sun.com/developer/onlineTraining/Beans/beans02/.



김 병 준

e-mail : windjun@kebi.com

2000년 대구효성가톨릭대학교 전자정보공학부 졸업(이학사)

2002년 대구가톨릭대학교 전산통계학과 전자계산학전공(이학석사)

2002년~현재 (주)라스21 연구개발부

관심분야 : 객체지향 소프트웨어공학, 재사용, 설계패턴, 컴포넌트기술

김 지 영

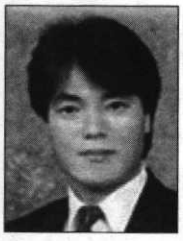
e-mail : kimjy@cataegu.ac.kr

1997년 대구효성가톨릭대학교 전자계산학과 졸업(이학사)

2000년 대구효성가톨릭대학교 전산통계학과 전자계산학전공(이학석사)

2000년~현재 대구가톨릭대학교 전산통계학과 전자계산학전공 박사수료

관심분야 : 객체지향 소프트웨어공학, 재사용, 설계패턴, 컴포넌트기술



김 행 곤

e-mail : hangkon@cataegu.ac.kr

1985년 중앙대학교 전자계산학과 졸업(공학사)

1987년 중앙대학교 대학원 전자계산학과 졸업(공학석사)

1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)

1978년~1979년 미 항공우주국 객원연구원

1987년~1989년 AT&T 객원연구원

2000년~2002년 미 센트럴 미시건대학 교환 연구 교수

1990년~현재 대구가톨릭대학교 컴퓨터정보통신공학부 부교수

관심분야 : 객체지향 소프트웨어공학, 재사용, 설계패턴, 컴포넌트기술