자바 프로그램의 재사용을 위한 자바 빈즈 컴포넌트의 추출 및 명세화 기법

이 성 은[†] · 김 영 익[††] · 류 성 열[†††]

요 약

최근 소프트웨어와 시스템 개발의 중요한 기술적 변화는 조립 가능하고, 독립적으로 추출된 컴포넌트들의 조합에 의해 만들어지는 시스템을 제공하는 것이다. 컴포넌트 기반 소프트웨어의 주요한 장점은 개발 기간의 단축과 비용 감소에 있다. 이는 새로운 컴포넌트를 개발할 때 보다 이미 만들어진 컴포넌트를 사용하는 개발에서 더 효율적이다.

컴포넌트 기반의 소프트웨어 개발에는 두 가지 방법이 있다. 하나는 컴포넌트를 자체 개발하는 것이고, 다른 하나는 제3자가 개발한 컴포넌트를 구입하여 사용하는 것이다. 컴포넌트를 개발하는 경우, 기존의 컴포넌트화 되지않은 프로그램도 컴포넌트 개발에 재사용 할 수 있어야 한다.

본 논문은 자바 프로그램의 재사용성을 높이기 위해 두 가지 방법으로 접근한다. 첫째, 기존에 존재하는 자바 프로그램으로부터 자바 빈즈 컴포넌트 모델에 적합한 구성 요소를 추출하는 방법을 제시하고, 추출된 구성 요소를 중심으로 빈즈 컴포넌트로 변환하기 위한 프로세스와 기법을 제시한다. 둘째, 기존의 자바 프로그램이나 자바 빈즈 컴포넌트로부터 자동적으로 컴포넌트 설계 명세 정보를 추출한다. 추출된 정보를 XML로 표현하여 컴포넌트 환경의 재사용성을 높일 수 있다.

Extraction and Specification Technique of Java Components for Reuse of Java Programs

Sung-Eun Lee[†] · Young-Ick Kim[††] · Sung-Yul Rhew[†††]

ABSTRACT

An important technical issue in recent software development is to make needed software by the composition of components that are assemblable, and configurable, and independently extracted. The main advantage of component-based software development is reducing development time and cost. It is more cost-effective in development time to use components that are already developed than developing from scratch.

There are two ways of component-based software development : one is to compose self-developed components, and the other is to buy the components developed by third-parties and compose them. In the second case, existing non component programs must can be used for reuse in the component development.

In this paper, we approach two methods for increase of reusability of Java program. First, we suggest the technique of extracting the elements suitable for the Beans component model from Java program, and then we show a process and a guidline of converting the extract elements into the Beans component model. Second, we suggest a technique of automatically extracting component information from the Java Beans component, expressing them in XML, it is possible to reuse the efficient component environment.

† 정 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 정 회 원 : 코넷 엔지니어링
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
논문접수 : 2000년 2월 10일, 심사완료 : 2000년 4월 6일

1. 서 론

최근 소프트웨어 개발 추세의 특징은 소프트웨어의 대규모화와 짧은 생명 주기로 나타난다. 대규모의 소프트웨어를 단기간에 개발한다는 것은 기존 소프트웨어의 재사용과 유지 보수가 없이는 어렵게 된다. 이를 위해서는 전반적인 소프트웨어의 설계나 원시 코드의 생성 등에서 유지 보수를 위한 소프트웨어의 이해는 물론, 컴포넌트화 되어 있지 않은 프로그램을 컴포넌트화 함으로써 재사용성을 높여주는 일 등이 필요하다. 컴포넌트 기반의 소프트웨어의 장점은 개발 기간의 단축과 개발 비용의 감소 등을 들 수 있다[8, 13].

지금까지 재사용을 위해서는 재사용을 위한 컴포넌트 자체를 새롭게 개발하고, 일반 어플리케이션에 대하여는 기존 소프트웨어의 소스코드 차원에서의 재사용이 고려되었다. 그러나 이러한 경우, 많은 자원의 낭비 뿐만 아니라 최악의 경우 재사용을 고려한 소프트웨어를 처음부터 새롭게 개발해야 한다는 문제점이 있다. 또한 이미 개발된 컴포넌트를 사용함으로써 얻는 개발 기간 단축은 컴포넌트나 모듈을 자체 개발 하는 경우보다 외부로부터 구입하는 경우에 비용면에서 효율적일 수 있으나, 일반적인 컴포넌트들이 대부분 바이너리 파일 형태로 공급되므로, 이런 상황에서는 어떻게 원하는 기능을 추가하고 향상시키기 위한 유지보수를 할 것인가를 고려해야 한다.

현재 CASE도구들 중에는 원시 코드로부터 문서화가 가능한 것들이 있으나, 바이너리 컴포넌트 파일로부터 문서화를 지원하는 CASE 도구는 존재하지 않는다. 게다가 지원하는 문서의 형식이 일반적인 텍스트 파일이나 HTML 파일이므로 다음과 같은 여러 문제점들이 있다[9, 10].

- 각 도구마다 서로 다른 문서 형식을 지원하므로 도구 간의 정보 교환이 불가능하다.
- 문서의 형태를 구조적으로 정의하거나 확장이 불가능하다.
- 문서 자체가 유효한지 검증의 어려움이 있다.

이러한 문제점들은 XML 형식으로 문서화 함으로써 극복할 수 있다[10].

본 논문은 프로그램의 재사용을 위해 이미 존재해 있는 어플리케이션을 재사용 가능한 컴포넌트로 변환하는 기법을 제시하고, 기존 컴포넌트 명세화의 문제점을 해결하기 위하여 XML을 기반으로 한 컴포넌트 명세를 제안한다.

본 논문에서는 자바 프로그램과 자바 빈즈(Java Beans) 컴포넌트를 적용 모델로 한다. 자바 빈즈는 자바 언어를 기반으로 한 컴포넌트 모델로, 소스 코드 없이 클래스 파일만을 이용하여 컴포넌트의 속성, 메소드, 이벤트 등의 정보를 얻을 수 있다[2, 13, 15]. 이 정보들을 이용하여 컴포넌트의 재사용과 유지 보수를 위한 명세를 얻고자 한다.

2. 관련연구

본 장에서는 논문과 관련된 일반 사항에 대한 연구로, 자바 빈즈 컴포넌트와 XML에 대한 특성들을 보인다.

2.1 자바 빈즈 컴포넌트

자바 빈즈는 Sun에서 제시한 자바 언어를 위한 소프트웨어 컴포넌트 모델이다. 자바 빈즈는 자바 언어의 장점을 그대로 살려 이식성이 있고 플랫폼에 독립적이며, 재사용 가능한 소프트웨어 컴포넌트로서 개발 도구에서 시각적으로 조작이 가능하다. 자바 빈즈 컴포넌트가 갖는 특징은 다음과 같은 것들이 있다[12, 13].

- 속성(Properties) : 자바 빈은 소프트웨어 컴포넌트로서 자신만의 속성을 갖는다.
- 이벤트(Events) : 자바 빈에서 이벤트 모델이 컴포넌트들간의 통신을 위해서 사용된다. 이벤트를 발생하는 이벤트 소스(Event source)와 이벤트를 받는 이벤트 리스너(Event listener)가 있다.
- 지속성(Persistence) : 빈의 상태를 영구적으로 저장하는 것을 말한다.
- 내부 투영(Introspection) : 빈의 역할이 무엇인지를 알게 한다. 빈의 개발자가 빈을 설명하는 BeanInfo 클래스를 제공하거나 자바 내부 검사(reflection)를 이용하는 두 가지 방법이 있다. 본 논문에서는 이러한 내부 투영이나 내부 검사를 이용하여 자바 빈즈 컴포넌트로부터 명세 정보를 추출한다.
- 커스터마이제이션(Customization) : 일반적으로 자바 빈을 사용할 경우 직접 코드 작성을 하여 빈을 제어할 수도 있지만 어플리케이션 개발 도구에서 시각적으로 사용자의 목적에 맞게 설정할 수 있다.

2.2 XML(eXtensible Markup Language)

XML은 1996년 W3C에 의해 발표된 SGML의 부분 집합으로 기존의 웹 언어인 SGML과 HTML의 여러 단점을 보완한다[4, 16]. XML은 자신의 마크업(markup) 요소들을 정의할 수 있기 때문에 확장이 가능하고 유연성, 다중 링크 등의 특징을 갖는다[4, 14]. 따라서 원하는 문서를 구조적으로 표현하는 것이 가능하다. XML 문서에서 사용할 태그를 정의하고, 이들이 어떤 순서로 동작하며, 어떤 태그가 다른 태그를 포함하는지 정의할 수 있다. 또한, 그 문서의 문서 타입 선언(Document Type Declaration)을 지정할 수 있다. DTD는 많은 XML 문서에서 반드시 필요한 것은 아니지만, XML이 규칙에 맞는지 확인하기 위해서는 DTD를 반드시 포함해야 한다. 또한 XSL은 XML 문서를 원하는 형태로 변형시켜 사용자에게 제공하는 역할을 한다. XSL이 없을 경우, 문서는 XML의 태그를 포함하여 보여주지만 XSL을 사용하면 HTML처럼 태그는 보이지 않고 원하는 데이터만 볼 수 있게 해준다[4, 14].

2.3 XML을 이용한 컴포넌트의 명세

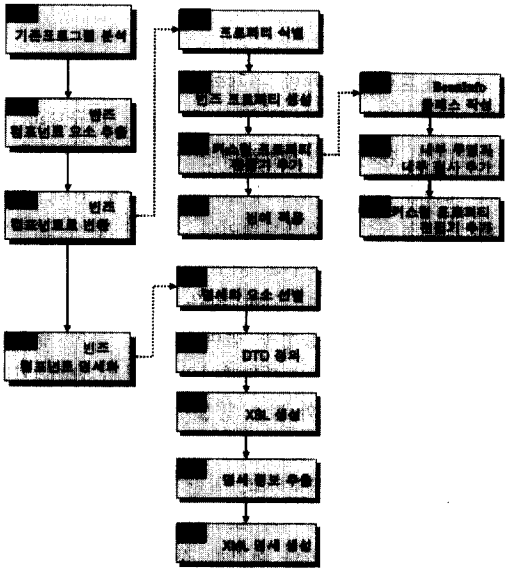
현재 컴포넌트 분야에서의 XML을 이용한 명세는 EJB 1.1 spec에 명세 되어 있다.

EJB 컴포넌트는 클래스 파일들을 jar로 패키지화하여 배포되는데, 이때 배포할 컴포넌트에 대한 정보를 XML 파일로 명세하게 되어 있다. EJB XML 명세는 서버 측의 컴포넌트를 명세화 하는데 사용된다. 그러나 EJB에서의 DTD가 방대하여 일반 사용자가 사용하기에는 어려움이 있고, 표준화된 명세를 바꾸어 사용한다는 데에도 문제가 있어 이에 대한 개선이 필요하다.

본 논문에서는 클라이언트에서 사용되는 자바 빈즈 컴포넌트, 컴포넌트가 아닌 일반 자바 프로그램의 클래스 파일이 모두 명세가 가능하도록 하고, 사용자가 원하는 정보를 표시하기 위하여 DTD를 수정하도록 제안한다. 또한 DTD에는 명세에 필요한 최소한의 정보만을 포함하도록 정의하여 사용자들이 간단히 사용할 수 있도록 제안한다.

3. 컴포넌트 변환-명세화 프로세스

본 논문은 자바 프로그램에서 컴포넌트로의 변환과 명세화를 위해서 (그림 1)과 같은 단계들을 갖는 프로세스를 제안한다.



(그림 1) 컴포넌트 변환-명세화 프로세스

3.1 기존 프로그램 분석

이 단계에서는 이미 존재해 있는 컴포넌트로 변환될 대상 프로그램으로부터 인터페이스, 클래스, 상속관계, 메소드, 이벤트, 그리고 프로퍼티 등의 정보를 분석한다.

3.2 빈즈 컴포넌트 요소 추출

일반적으로 자바 빈즈 컴포넌트는 그래픽 요소(UIE : User Interface Elements)를 가지고 있는 클라이언트측의 재사용 컴포넌트지만 UIE를 가지지 않는 컴포넌트도 있다. 이런 특징을 갖고 있는 자바 빈즈 컴포넌트로 자바 프로그램을 변환하기 위해서는 자바 프로그램의 유형을 구분해야 하고, 유형별로 구분된 프로그램을 분석하여 자바 빈즈 다음의 유형에 맞는 컴포넌트 요소들을 추출해야 한다. 자바 빈즈의 유형은 <표 1>과 같다.

3.2.1 가시적 자바빈으로부터 컴포넌트 요소 추출

클래스(수퍼 클래스 포함), 인터페이스, 생성자, 메소드, 이벤트, 파라미터, GUI객체의 메소드(빈의 프로퍼티와 관계될 수 있고 BeanInfo에 의해 정보 핸들링이 가능하며 커스텀 프로퍼티 편집기 작성시의 커스텀 프로퍼티가 될 수 있다.)

3.2.2 비가시적 자바빈으로부터 컴포넌트 요소 추출

클래스(수퍼 클래스 포함), 인터페이스, 생성자, 메소

드, 이벤트, 파라미터를 추출한다.

〈표 1〉 자바 빈즈의 유형

유 형	종류 및 특성
가시적 자바빈	컴포넌트(Component), 컨테이너(Container) 클래스에 기반을 둠
비가시적 자바빈	컴포넌트 클래스에 기반한 클래스에서 상속된 것이 아니므로 보이지 않음 데이터베이스 연결, 필터 또는 알고리즘, 그리고 속성을 가지거나 위임형 이벤트 모델을 따르는 클래스
애플릿 자바빈	애플릿 샐행환경을 만들기 위해 빌더 툴은 AppletContext 인터페이스를 확장한 클래스를 생성함 AppletStub 인터페이스를 구현하는 컨테이너 클래스 생성함 AppletStub 인터페이스는 Applet 객체를 유지하고, 애플릿의 변수값(<applet> 태그 내의 <param>값)들을 저장함

3.2.3 애플릿 자바빈으로부터 컴포넌트 요소 추출

클래스(수퍼 클래스 포함), 인터페이스, 생성자, 메소드, 이벤트, 파라미터, HTML 파일의 <applet> 태그 내의 <param> 태그에 명시되어 있는 파라미터 이름과 값(빈의 프로퍼티가 될 수 있다.)을 추출한다.

3.3 컴포넌트 변환

자바 프로그램이 자바 빈즈 컴포넌트 모델로 변환되기 위해서는 프로그램의 구조를 자바 빈즈에 맞게 수정하거나 변경해야 하고, 자바 프로그램으로부터 프로퍼티를 식별해야 한다.

먼저 자바 빈즈 구조에 맞게 수정하는 과정에서는 생성될 자바 빈즈 클래스가 Serializable interface를 구현해야 하고, 모든 비활성 변수들을 transient로 표시해야 한다. 그리고 자바 빈즈는 매개변수가 없는 생성자를 필요로 하기 때문에 생성할 자바 빈즈 컴포넌트에 반드시 매개 변수 없는 생성자를 추가한다.

3.3.1 프로퍼티 식별

프로퍼티는 클래스가 어떻게 보여지고 어떻게 동작하는지를 클래스 사용자가 커스터마이즈 하도록 허용한다. 자바 프로그램의 속성(attributes)은 프로퍼티가 아니다. 그러나 자바 프로그램의 속성이 새로운 빈즈의 프로퍼티가 될 후보일 수는 있다. 자바 빈즈 프로퍼티는 클래스의 몇 가지 내부 상태를 검색하거나 변경할 수 있는 메소드로 구성되어 있다.

자바 어플리케이션에서는 Component, Container를 상속하는 클래스에 의해 생성된 객체 각각의 속성은 빈즈의 프로퍼티가 될 수 있다.

자바 애플릿의 매개변수는 빈즈의 프로퍼티가 될 수 있고 모두 문자열 타입이다. 빈즈의 많은 프로퍼티들은 애플릿의 이런 문자열 표현으로부터 몇몇 다른 타입으로의 형 변환(type conversion)이 필요하다.

3.3.2 빈즈 프로퍼티 생성

프로퍼티를 생성하기 위해서는 빈즈 클래스 코드를 만들어야 하는데, 본 연구에서는 원시코드의 호환성과 가용성을 고려하여, 존재해 있는 클래스(또는 애플릿)의 서브클래스를 만들고 그것에 빈즈 정보를 추가하는 방법을 이용한다.

식별된 각 프로퍼티가 Simple 프로퍼티, Indexed 프로퍼티, Bound 프로퍼티, 그리고 Constrained 프로퍼티인지를 파악해서 해당 프로퍼티의 접근 메소드를 추가하여 프로퍼티를 생성한다. 프로퍼티의 생성은 Sun에서 정의한 'Design Patterns for Properties'를 따른다[6].

1) Simple 프로퍼티 : 특별히 객체 내의 공용 변수를 참조할 때 단순 프로퍼티를 사용하도록 한다. "get-<PropertyName>"과 "set<PropertyName>" 메소드는 동일한 형(type)을 반환하고, 읽기-쓰기 프로퍼티를 정의하기 위한 메소드의 이름은 "<PropertyName>"이 된다. "get<PropertyName>" 메소드는 프로퍼티의 값을 얻는데 사용되고 "set<PropertyName>" 메소드는 프로퍼티의 값을 설정하는데 사용되어진다. 이 두 개의 메소드는 동일한 하나의 클래스 내에 위치하거나, 또는 하나는 베이스 클래스에 위치하고 다른 하나는 유도된 클래스 내에 위치할 수 있다.
2) Indexed 프로퍼티 : 인덱스 프로퍼티는 값의 범위를 제공한다. 빈즈 개발자가 특별하게 정하는 인덱스 값에 의해 프로퍼티를 읽고 쓸 수 있다. 프로퍼티 인덱스의 형(type)은 정수형(int)이어야 하고, 일반적으로 배열, 벡터, 또는 목록 유형의 프로퍼티를 접근하는데 사용하도록 한다. 인덱스된 getter, setter 메소드가 참조하는 인덱스 값이 현재 배열의 범위를 벗어나면 예외를 발생시킨다. 배열의 크기를 변경하고자 할 때는 배열 setter 메소드를 이용해서 값을 변경해야 한다.

3) Bound 프로퍼티 : 연결된 프로퍼티는 변화될 때마다 그 값을 방송해야 하는 프로퍼티이므로 연결된 프로퍼티는 변화가 일어날 때마다 등록된 리스너에게 통보해야 한다. 그래서 클래스 사이에서 상태 변화를 전달하고자 할 때 연결된 프로퍼티를 사용하도록 한다.
연결된 프로퍼티는 리스너가 프로퍼티 변화를 관리하기 위해 addPropertyChangeListener()와 removePropertyChangeListener() 메소드를 갖는다.

4) Constrained 프로퍼티 : 연결된 프로퍼티처럼 프로퍼티의 변경을 감지하지만 제한된 프로퍼티에 등록된 리스너가 변경을 거부할 수 있다. 자바빈에서 제한된 프로퍼티의 setter 메소드는 PropertyVetoException의 제공을 요구하게 된다. 제한된 프로퍼티 상의 임의의 프로퍼티가 바뀌면 등록된 특정 리스너 상의 VetoableChangeListener. vetoableChange 메소드를 호출해야 하고, 전달되는 PropertyChangeEvent 객체는 프로퍼티의 지역 독립적 이름과 바뀌기 전과 바뀐 후의 새로운 값을 은닉한다.
내부 투영(Introspector) 클래스는 클래스의 프로퍼티를 자동적으로 식별하기 위해 위에서 기술한 내용을 토대로 생성된 프로퍼티를 찾는다.

3.3.3 커스텀 프로퍼티 편집기 추가

3.3.3.1 BeanInfo 클래스 작성

실행 시와 빌더 환경 안에서 자바 빈즈가 제공하는 프로퍼티, 이벤트, 그리고 메소드들이 수행될 필요가 있는데, 이것을 내부 검사라고 한다. 내부 투영을 사용해 빈 안에 있는 메소드를 찾아내고 이들을 프로퍼티 접근, 이벤트 발생, 기타 메소드 등의 범주로 분류하는 기능을 수행하는 것이 인트로스펙터 클래스이다. 이 인트로스펙터 클래스에 의해 발견된 정보들은 BeanInfo 객체에 저장된다.

이런 내부 검사 기능을 제공하기 위해서는 프로퍼티를 담고 있는 자바 빈즈 클래스에 대해 BeanInfo 인터페이스를 표현하는 클래스를 만들어야 한다. BeanInfo는 어플리케이션 빌더 도구가 이 빈을 다른 컴포넌트와 연결시키고 프로퍼티를 변경하는데 사용된다. BeanInfo 클래스에서는 BeanInfo 인터페이스의 여러 메소드들을 구현해 빈의 정보를 얻을 수 있다.

BeanInfo 클래스를 만들기 위해서는 자바 빈 클래스 이름에 "BeanInfo"라는 문자열이 덧붙혀진 형태를 가져야 한다.

BeanInfo에는 BeanDescriptor, EventSetDescriptor, FeatureDescriptor, IndexedPropertyDescriptor, PropertyDescriptor, MethodDescriptor, ParameterDescriptor 등 7개의 디스크립터를 통해 빈으로부터 여러 정보를 얻는다. BeanDescriptor에서는 보여질 이름과 커스터마이저 클래스와 같은 빈 자신에 관한 자세한 정보를 얻을 수 있고, EventSetDescriptor에서는 빈에 의해 발생되어지는 사건의 이름, 속성, 그 외 여러 정보들을 얻을 수 있으며, PropertyDescriptor, MethodDescriptor에서는 빈에 의해 드러나는 프로퍼티 및 메소드의 이름, 속성과 여러 정보들을 얻을 수 있다.

BeanInfo 객체는 다음의 세 가지 방법을 이용해 만든다.

1) 내부 투영을 통해 최소한의 정보를 채우기 위해 인트로스펙터 클래스를 사용
2) BeanInfo를 구현하는 클래스를 빈즈 개발자가 직접 작성
3) BeanInfo 객체의 특정한 부분에 대한 개발자의 정보와 인트로스펙터 클래스를 결합시키고 남은 데이터를 얻기 위해 코어 내부 투영을 사용

3.3.3.2 내부 투영과 내부 검사 추가

1) 내부 투영

내부 투영은 클래스 검사, 객체 조작, 배열 작업 등 크게 세 범주로 나뉘어진다[12]. 각 범주별로 수행해야 할 작업은 다음과 같다.

① 클래스 검사(Examining Classes) : 객체의 클래스를 어떻게 결정할 것인가와 클래스, 인터페이스에 관한 정보를 어떻게 얻을 것인가에 대해 검사한다. 즉, 클래스 객체 검색, 클래스 이름 얻기, 클래스 수정자 발견, 수퍼 클래스 찾기, 클래스에 의해 표현되어지는 인터페이스 식별, 인터페이스 시험, 클래스 필드 식별, 클래스 생성자 발견, 그리고 메소드 정보 획득 등의 기능을 수행한다.

② 객체 조작(Manipulating Objects) : 어떻게 클래스를 객체화하고, 필드값을 얻고 설정할 것인가, 어떻게 메소드를 요청할 것인가 등을 보인다. 실행 시간까지 클래스, 필드, 메소드의 이름을 모르

는 경우에도 위의 작업을 실행할 수 있다. 구체적으로 객체 생성, 필드값 얻기, 필드값 설정, 그리고 메소드 요청 등의 기능을 수행한다.

③ 배열 작업(Working with Arrays) : 실행시간까지 알려지지 않은 배열을 생성하거나 변경한다. 구체적으로 배열 식별, 컴포넌트 타입 검색, 배열 생성, 그리고 배열의 원소값을 얻어오거나 설정하는 기능을 수행한다.

2) 내부 검사

인스펙터 클래스가 코어 내부 투영 API를 이용해 빈을 검사하고 정보를 얻는다.

3.3.3.3 커스텀 프로퍼티 편집기 추가

IBM의 Visual Age, Lotus의 BeanMachine, Sun의 BDK(Bean Development Kit)에 들어있는 BeanBox 등의 툴들은 자체적으로 대부분의 표준 프로퍼티 타입인 int, boolean, String, java.awt.Color 등을 제공한다. 그러나 java.util.Locale 또는 java.net.URL 등과 같은 타입의 지원은 부족하기 때문에 실제적으로 자바 빈즈 개발자들은 이런 표준 프로퍼티만을 이용해 빈즈를 만들지 않는다. 오히려 그들의 필요에 의해 대부분 자신들의 프로퍼티를 정의해서 사용한다. 이런 이유로 자바 프로그램으로부터 프로퍼티를 식별하는 것은 어려운 과정이다. 이 문제를 해결하기 위해 자바 빈즈의 PropertyEditor 인터페이스를 사용한다[13]. 이 인터페이스는 java.beans 패키지(package) 안에 위치해 있으며, 어떠한 빈 빌더 툴의 프로퍼티 시트와도 통합될 수 있다.

커스텀 프로퍼티 편집기는 BDK의 BeanBox 처럼 IDE와 빈 지원 응용 프로그램을 확장한다. 커스텀화가 없다면 빈 지원 IDE에서 편집할 수 있는 프로퍼티는 지원되는 클래스 유형에 한정될 수 밖에 없기 때문에 빈이 어느 IDE에 인스톨될 것인가에 구애 받지 않고 사용될 수 있도록 빈의 부분을 변경하는 편집기를 제공해야 한다. 커스텀 프로퍼티 편집기를 작성하려면 PropertyEditor 인터페이스를 구현해야 한다.

물론 프로퍼티 편집을 위해 커스텀 프로퍼티 편집기를 별도로 만들 필요는 없다. 그러나 커스텀 프로퍼티 편집기를 별도로 만듦으로 해서 프로퍼티 편집기가 직접 빈과 통신하지 않고 IDE와 통신하도록 할 수 있고, 최종 빈에 배포되는 편집을 위한 코드가 감소되는 잇점이 있다. 빈의 프로퍼티를 편집하기 위해서는 PropertyEditor 인터페이스의 getValue() 메소드와 setValue() 메소드를 커스텀 프로퍼티 편집기에서 구현해 주어야 한다. 이것은 빈 개발자에게 또 다른 코드 작성의 부담을 안겨주기 때문에 PropertyEditor의 여러 메소드를 일반적으로 미리 정의해 놓고 커스텀 프로퍼티 편집기에서는 상속해서 사용하기만 하도록 정의된 것이 있는데 그것이 PropertyEditorSupport 클래스이다.

(그림 2)와 (그림 3)은 PropertyEditor 인터페이스와 PropertyEditorSupport 클래스를 이용하는 각각의 경우에 대한 예를 보여준다.

```
Public class MyPropertyEditor implements PropertyEditor {
Private MyProperty myprop;

Public void setValue(Object value) {
Myprop = (MyProperty)value;
// Handle the property change if nessary
. . .
}

public Object getValue() {
return myprop;
}
. . .
}
```

(그림 2) PropertyEditor 인터페이스를 구현하는 커스텀 프로퍼티 편집기

```
Import java.math.*;

Public class BigIntegerEditor extends java.beans.PropertyEditorSupport {
Public String getAsText() {
    return getValue().toString();
}

public void setAsText(String text) throws IllegalArgumentException {
SetValue(new BigInteger(text));
  }
}
```

(그림 3) PropertyEditorSupport를 상속하는 커스텀 프로퍼티 편집기

빈의 프로퍼티를 편집하는 방법에는 텍스트에 의한 방법, 컴포넌트에 의한 방법, 그리고 열거형(enumeration)에 의한 방법 세 가지가 있다. 그러나 대부분의 빈 빌더 툴들은 텍스트에 의한 프로퍼티 편집 방법만을 제공하기 때문에 빈 개발자가 별도로 컴포넌트나 열거형에 의한 프로포티 편집 기능은 불필요하다.

getAsText()와 setAsText()는 텍스트 방법으로 프로퍼티를 핸들링할 때 사용되는 메소드이다. 이들 메소드를 이용해 커스텀 프로퍼티 편집기를 구현하고자 할 때에는 다음의 사항을 기억해야 한다.

1) setAsText() 메소드로 잘못된 사용자 입력이 들어오는지를 검사해서, 잘못이 발견되면 IllegalArgumentException 예외를 발생시킨다.
2) 어떠한 예외 발생 없이 getAsText() 메소드가 반환하는 어떤 값도 핸들링 할 수 있도록 setAsText() 메소드를 정확하게 만들어야 한다.
3) 프로퍼티 데이터를 위해 getAsText()와 setAsText() 메소드는 아주 간단하게 해서 사용해야 한다.

프로퍼티 편집기에서 텍스트와 컴포넌트에 의한 프로퍼티 편집을 지원하더라도 대부분의 빌더 툴은 텍스트 방식의 프로퍼티 편집 기능을 제공하므로 getAsText()와 setAsText() 메소드를 구현할 때 컴포넌트 방식의 프로퍼티 편집 기능을 구현할 필요가 없다.

3.3.4 빈즈 적용

컴포넌트로 변환될 빈이 다른 객체의 프로퍼티가 변화되는 것을 듣는 사건 수신자가 되도록 java.beans.PropertyChangeListener 인터페이스를 구현한다. 이와 비슷하게 변환될 빈즈 컴포넌트가 그 안에 addPropertyChangeListener()와 removePropertyChangeListener() 메소드를 구현함으로써 사건을 발생시킬 수 있게 한다. 이것은 이들 두 메소드 내에서 PropertyChangeSupport 클래스의 인스턴스로써 addPropertyChangeListener()와 removePropertyChangeListener() 메소드를 호출하게 함으로써 가능하다. 이렇게 함으로써 변환될 빈 컴포넌트는 다른 객체에 의해 변경되어지는 자신의 프로퍼티를 인지하고, 이전의 프로퍼티값과 새로운 값을 보전할 수 있다. 이 때에는 Bound 프로퍼티나 Constrained 프로퍼티를 이용해서 빈 컴포넌트와 객체를 연결해준다.

3.4 빈즈 컴포넌트 명세화

3.4.1 명세화 요소 선별

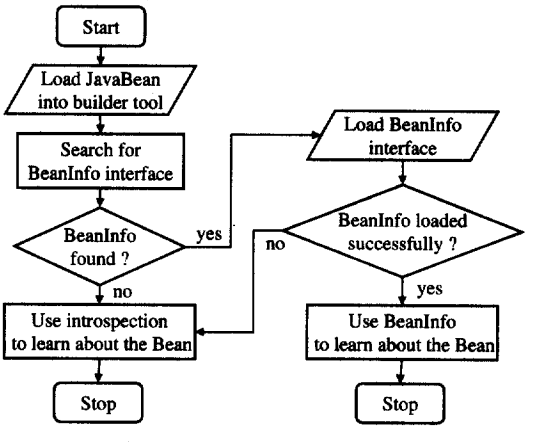
자바 빈즈 클래스는 일반적인 자바 클래스와 동일한 구조를 갖는다. 자바 빈즈 컴포넌트로부터 추출 가능한 요소 선별을 위해서는 일반적인 자바 클래스와 자바 빈즈 컴포넌트 클래스로부터 정보를 얻어내는 방법을 모두 고려한다. 일반 자바 클래스의 경우, 내부 검사를 이용하여 클래스의 정보를 얻을 수 있다. 자바 빈즈 클래스도 일반 자바 클래스와 동일하므로 내부 검사를 이용하여 정보 추출이 가능하나 빈즈 자체가 갖는 특성을 지원하기 위해 내부 투영을 이용할 수도 있다.

3.4.1.1 내부 검사를 이용한 추출

자바 클래스와 자바 빈즈 클래스에서 내부 검사(reflection)를 이용하여 정보를 추출할 수 있다. 내부 검사 API는 자바가 클래스 파일로부터 직접 클래스와 메소드 정의를 동적으로 읽을 수 있는 클래스의 집합이다. 본 논문에서는 이들 요소들 중에서 class name, super classes, interfaces, methods만을 채택한다.

3.4.1.2 내부 투영을 이용한 추출

자바 빈즈 클래스에 대해서는 자바 빈즈의 특성에 맞도록 좀 더 유연한 기능을 가진 내부 투영(introspection)을 사용할 수 있다.



(그림 4) 내부 투영을 이용한 자바 빈즈로부터 명세 정보를 얻는 순서도

내부 투영에서 자바 빈즈 클래스가 BeanInfo 클래스를 갖는 경우는 BeanInfo를 이용하며, BeanInfo 클래스를 갖지 않는 경우는 내부 검사를 이용한다.

3.4.2 DTD(Document Type Declaration) 정의

XML로 자바 빈즈 컴포넌트를 표현하는 이유는 속성-값의 쌍으로 연관되는 문서의 논리적 구조와 저장

레이아웃 기법을 XML이 제공하기 때문이다. 바로 이러한 기법을 DTD라고 하며 논리적 구조에 대한 제한사항을 정의한다[14].

자바 빈즈 컴포넌트 명세를 XML로 표현했을 때, 위에서 표현된 자바 빈즈 클래스 구성 요소들이 XML 문서에 포함된다. 이때 표현된 XML 문서가 유효한지 자체 검증이 가능하도록 하려면 DTD를 가져야 한다. DTD가 정의된 XML 파일은 논리적으로 유효하고 일관성을 가지므로 일정한 형식에 맞도록 표현과 자체 검증이 가능하다. DTD를 정의하기 위해서는 추출된 자바 빈즈 컴포넌트 구성 요소들과 구성 요소들의 관계를 파악해야 한다.

DTD는 XML 파일의 내부와 외부에 또는, 양쪽 모두 둘 수 있다. 본 논문에서는 한번 정의된 DTD가 여러 빈들의 명세에 똑같이 사용될 수 있으므로 외부 DTD를 정의해서 사용하기로 한다. 정의된 DTD는 (그림 5)와 같다.

```
<!ELEMENT document (bean)>
<!ELEMENT bean (className,superClasses,interfaces,properties,methods,events)>
<!ELEMENT className (#PCDATA)>
<!ELEMENT superClasses (superClass)*>
<!ELEMENT superClass (#PCDATA)>
<!ELEMENT interfaces (interface)*>
<!ELEMENT interface (#PCDATA)>
<!ELEMENT properties (property)*>
<!ELEMENT property(propertyType,propertyName,readMethod,writeMethod)>
<!ELEMENT propertyType (#PCDATA)>
<!ELEMENT propertyName (#PCDATA)>
<!ELEMENT readMethod (#PCDATA)>
<!ELEMENT writeMethod (#PCDATA)>
<!ELEMENT methods (method)*>
<!ELEMENT method (#PCDATA)>
<!ELEMENT events (event)*>
<!ELEMENT event (listenerType,listenerMethods,addListenerMethod,removeListenerMethod)>
<!ELEMENT listenerType (#PCDATA)>
<!ELEMENT listenerMethods (listenerMethod)*>
<!ELEMENT listenerMethod (#PCDATA)>
<!ELEMENT addListenerMethod (#PCDATA)>
<!ELEMENT removeListenerMethod (#PCDATA)>
```

(그림 5) 자바 빈즈 컴포넌트 명세의 DTD

3.4.3 XSL 생성

XSL은 XML 문서와 데이터의 포맷팅 정보를 기술하기 위해 개발된 스타일 쉬트 언어(Style sheet language)이다. 문서의 내용과 포맷의 정보를 독립된 형태로 저장하도록 스타일 쉬트를 표준화하면 문서를 만들고 관리하는 모든 소프트웨어의 상호 호환성을 증진시킬 수 있다. XSL 파일이 없이 XML 파일만을 브라우저로 보게 되면, 데이터 뿐만 아니라 태그까지 보이게 된다. 따라서 사용자에게 원하는 문서 데이터만을 적절한 형태로 제시하기 위해서는 XSL을 생성하여야 한다. (그림 6)은 본 논문에서 사용될 XSL의 일부분이다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<head>
<title>JavaBeans XML Specification</title>
</head>
<body>
<div style="text-align:left">
</div>
<xsl:apply-templates select="/bean/className"/>
<h4><font color = "blue">Super classes</font></h4>
. . .
</xsl:template>
</xsl:stylesheet>
```

(그림 6) 자바 빈즈 컴포넌트 명세를 위한 XSL

3.4.4 자바 빈으로부터 명세 정보 추출

자바 빈즈 컴포넌트에서 명세 정보를 추출하는 것은 먼저 getBeanInfo() 메소드에 의해 가능하다. 이 메소드는 클래스의 BeanInfo 객체를 먼저 검색함으로써 BeanInfo 객체를 만든다.

예를 들어 JTable라는 자바 빈즈 클래스에는 JTableBeanInfo라는 클래스를 만들 수 있다. 그리고 BeanInfo에 의해 제공되지 않는 정보를 위해 내부 투영을 사용하여 클래스를 분석한다. 만약 제공되는 BeanInfo 클래스가 없다면, getBeanInfo() 메소드는 이벤트, 메소드, 속성 디스크립터를 생성하기 위해 모든 이벤트, 메소드, 속성을 분석한다. 내부 투영으로 얻을 수 없는 정보는 내부 검사를 이용하여 얻게 된다.

3.4.4.1 내부 투영을 이용한 명세 정보 추출

다음과 같은 메소드들을 활용하여 필요한 명세 정보를 추출한다.

- getBeanInfo( ) 메소드 : 주어진 자바 빈을 내부 투영하고 빈의 모든 속성, 메소드, 이벤트를 조사한다.
- getPropertyDescriptor( ) 메소드 : 이 메소드는 빈이 구현하는 PropertyDescriptor 클래스의 배열을 리턴한다.

- getMethodDescriptors( ) 메소드 : 이 메소드는 빈이 구현한 MethodDesctiptor 클래스 배열을 리턴한다. 이 메소드들에는 get/set 속성 메소드와 add/remove 이벤트 메소드를 포함한다.
- getEventSetDescriptors( ) 메소드 : 이 메소드는 EventSet 객체의 배열을 리턴한다. 이벤트 집합은 add/remove 리스너 메소드와 그 이벤트 유형을 정의하는 데 사용된다.

3.4.2 내부 검사를 이용한 명세 정보 추출

1) 클래스의 이름 추출 : 모든 자바 클래스들은 하나의 이름을 갖는다. getName() 메소드를 이용하면 실행 시간에 동적으로 객체의 이름을 얻어올 수 있다.
2) 생성자 메소드 추출 : 클래스의 인스턴스를 만들기 위해서는 생성자 메소드를 호출하여야 한다. 생성자 메소드는 시그니쳐(signature)에 따라 여러 개의 생성자가 가능하다. 생성자 메소드를 얻기 위해서는 getConstructors() 메소드를 이용한다. 생성자 메소드를 구별하는 것은 파라미터와 파라미터의 타입이므로 파라미터의 정보도 함께 얻어와야 한다. 파라미터 정보를 얻기 위해서는 getParameterTypes() 메소드를 사용한다.
3) 수퍼 클래스 추출 : 자바 언어는 상속을 지원한다. 상속을 받는 서브 클래스는 수퍼 클래스의 모든 속성을 상속 받고 추가로 필요한 속성만을 정의하면 된다. 클래스를 설계하는 과정에서 어느 클래스로부터 상속을 받을지를 결정하는 것이 서브 클래스의 데이터와 기능을 결정하므로 중요할 수 있다. 따라서 어느 클래스로부터 상속을 받는지 알 수 있다면, 재사용과 유지 보수시에 커다란 도움이 될 수 있다. 어떠한 클래스가 상속 받는 수퍼 클래스를 알기 위해서는 getSuperclass() 메소드를 이용한다. 이 메소드의 리턴 값이 null 이면 상속 받는 수퍼 클래스가 존재 하지 않는다는 것이고, 계층적으로 상속 받는 수퍼 클래스를 모두 알고 싶으면 메소드의 리턴 값이 null일 때까지 반복적으로 호출하면 된다.
4) 인터페이스 추출 : 자바 언어는 C++와는 달리 하나의 수퍼 클래스로부터만 상속을 받을 수 있다. 그러나 자바 언어에서는 대신에 인터페이스라는 것을 지원한다. 인터페이스에서 메소드들을 선언하고 이를 구현하는 클래스에서 인터페이스 메소드의 내용을 채워서 사용하게 된다. 인터페이스도 수퍼 클래스에서처럼 클래스의 기능을 결정하므로 인터페이스 정보를 얻는 것이 필요하다. 인터페이스를 얻기 위해서는 getInterfaces() 메소드를 지원한다. 하나의 클래스에서 여러 개의 인터페이스를 구현하는 것이 가능하므로 이 메소드는 복수개의 인터페이스들을 리턴한다. 수퍼 클래스에서와 마찬가지로 반복적으로 사용하면 클래스가 구현하는 여러 인터페이스를 얻을 수 있다.

3.4.3 추출한 명세 정보를 XML로 생성

XML 파일은 빈에서 명세 정보를 추출하는 객체로부터 생성된다. 객체를 생성할 때 생성자 메소드 호출과 함께 전달된 자바 빈즈 클래스 파일의 이름을 입력으로 자바 빈즈 명세 추출 객체가 생성되는 것이다. 빈즈 클래스로부터 얻어진 정보들이 DTD에서 정의된 형식으로 XML 문서로 생성된다. 생성된 XML 파일은 일반 텍스트 에디터로도 볼 수 있으나, XML을 위한 스타일 쉬트 언어인 XSL을 지원하기 위해서는 XML과 XSL을 지원하는 브라우저를 이용하여야 한다. 현재 이를 지원하는 브라우저로는 MicroSoft사의 Internet Explorer 5.0이 널리 사용된다.

4. 사례연구

본 장에서는 3장의 (그림 1)에서 제시한 프로세스에 따라 자바 애플릿 프로그램을 자바 빈즈 컴포넌트로 변환하는 과정과, 기존의 바이너리 파일 형태의 빈즈 컴포넌트의 명세화 과정을 사례를 중심으로 실험한다.

4.1 컴포넌트 변환

(그림 7)은 변환 대상이 되는 색상 변화 기능을 수행하는 자바 애플릿 프로그램의 소스 코드이다.

(그림 7)로부터 추출 가능한 정보와 이 정보를 토대로 변환될 자바빈의 프로퍼티를 살펴보면 각각 <표 2>, <표 3>과 같다.

```
Public class MyPropertyEditor implements PropertyEditor {
Private MyProperty myprop;

Public void setValue(Object value) {
Myprop = (MyProperty)value;
// Handle the property change if nessary . . .
}

public Object getValue() {
return myprop;
}
. . .
}
```

```
public class ColorFadeBar extends Applet implements Runnable
{
protected int _iHpoints = 200;
protected int _iVpoints = 30;
protected int _ix = 0;
protected int _dy = 0;
. . .
protected int _iFadeDirection = LEFT;
protected int _iTextDirection = LEFT;
public void ColorFadeBar() { }
public void init()      {
int i;
Dimension dimSize = getSize();
_iHpoints = iGetParameter("WIDTH", dimSize.width);
_iVpoints = iGetParameter("HEIGHT", dimSize.height);
. . .
```

(그림 7) 색상 변화 애플릿 소스 코드

〈표 2〉 애플릿에 존재하는 매개변수

매개변수	의 미
_iHpoints	애플릿의 폭
_iVpoints	애플릿의 높이
_colorFrom	배경색의 시작 값
_colorTo	배경색의 끝 값
ColorText	나타낼 텍스트의 색
Ssring	나타낼 메시지 텍스트
_iFadeDirection	배경색의 색상변화 방향(left, right, up, down)
_iTextDirection	텍스트 정렬(left, center, right)
_messageFont	텍스트 폰트 이름, 텍스트 폰트의 크기
_ix	애플릿 모서리로부터의 텍스트 좌, 우 여백
_dy	텍스트 기준선 값(기본값 0)

〈표 3〉 생성된 자바빈의 프로퍼티

프로퍼티 이름	타입	내 용	매개변수
Height	Int	애플릿의 높이	HEIGHT
Width	Int	애플릿의 폭	WIDTH
Message	Sting	보여질 메시지	TEXT
MessageFont	Font	메시지의 폰트	FONT, FONTSIZE
TextDirection	Int	텍스트 정렬	TEXTDIR
MessageColor	Color	텍스트의 색	TEXTCOLOR
ColorFrom	Color	색상 변화의 시작 색	STARTCOLOR
ColorTo	Color	색상 변화의 마지막 색	ENDCOLOR
FadeDirection	Int	색상변화의 방향	FADEDIR

이렇게 식별된 프로퍼티들은 자바빈의 현재의 상태 정보를 얻거나 상태를 변경하기 위한 getter(), setter() 메소드가 된다. (그림 8)은 추출된 자바빈 프로퍼티가 자바빈에서 어떻게 사용되는지를 보여준다.

```
public void setWidth(int iWidth) { _iHpoints = iWidth; }
public int getWidth() { return _iHpoints; }
public void setHeight(int iHeight) { _iVpoints = iHeight; }
. . .
public void setMessageFont(Font font) {
if (font == null) {
  font = new Font("Helvetica", Font.BOLD, 16);
}
_messageFont = font;
}
public Font getMessageFont() {
  if (_messageFont == null) {
      return new Font("Sans-Serif", Font.BOLD, 16);
}
return _messageFont;
}
. . .
```

(그림 8) 변환된 빈의 getter(), setter() 메소드

이와 같이 빈이 수행할 기능을 기술한 후에는 빈 내부의 동작과 수행 흐름을 파악하기 위해 내부 투영(reflection), 내부 검사(introspection) 기능을 수행하는 BeanInfo 클래스를 작성한다.

```
public class ColorFadeBeanBeanInfo extends SimpleBeanInfo {
  protected PropertyDescriptor prop(String sName_, String sDesc_, boolean isBound_)
throws IntrospectionException {
  PropertyDescriptor pd = new PropertyDescriptor(sName_, ColorFadeBean.class);
  Pd.setShortDescription(sDesc_);
  Pd.setBound(isBound_);
  return pd;
  }
protected PropertyDescriptor prop(String sName_, String sDesc_, boolean isBound_, Class
classEditor_) throws IntrospectionException {
    System.out.println("Set editor for property " + sName_);
    PropertyDescriptor pd = prop(sName_, sDesc_, isBound_);
    pd.setPropertyEditorClass(classEditor_);
    return pd;
}
  public PropertyDescriptor[] getPropertyDescriptors() {
    try {
      PropertyDescriptor pds[] = {
        prop("Height", "Height", false),
        prop("Width", "Width", false),
        prop("Message", "The message to display", false),
. . .
        prop("FadeDirection", "The direction in which color fades",
            true, FadeDirectionEditor.class)
      };
      return pds;
    } catch (IntrospectionException e) {
        System.out.println("getPropertyDescriptors() Failed for " +
"ColorFadeBean: " + e.getMessage());
        return null;
      }
  }
};
```

(그림 9) 생성된 빈의 BeanInfo 클래스

빌더 툴에서 표준으로 제공하는 프로퍼티 편집기로는 각각의 빈의 프로퍼티를 적절하게 편집할 수 없기 때문에 커스텀 프로퍼티 편집기를 작성한다. 생성하고자 하는 ColorFadeBean의 커스텀 프로퍼티 편집기는 색상의 변화를 편집하기 위한 FadeDirectionEditor와 텍스트의 출력 방향을 바꾸기 위한 TextDirectionEditor 두 개가 있다. 이 중 (그림 10)에 TextDirectionEditor를 보인다.

```
Public class TextDirectionEditor extends java.beans.PropertyEditorSupport {
  public String[] getTags() {
    return new String[] { "Left", "Right", "Center" };
  }
public void setAsText(String sValue) throws IllegalArgumentException
{
    if (sValue == "Left")
      setValue(new Integer(ColorFadeBar.LEFT));
    else if (sValue == "Right")
    . . .
    throw new IllegalArgumentException(sValue);
  }
  public String getAsText() {
    int iValue = ((Integer)getValue()).intValue();
    if (iValue == ColorFadeBar.LEFT)
      return "Left";
      . . .
      else
        return "Illegal value";
}
. . .
};
```

(그림 10) 텍스트 변환 커스텀 프로퍼티 편집기

위와 같이 자바 프로그램을 자바 빈즈로 변환하였으면 컴파일하여 빌더 툴에서 실행 가능한 형태인 바이너리로 만들어야 한다. 이 바이너리 파일은 JAR 포맷을 갖는다. JDK 1.1에서부터 제공되는 JAR 파일은 클래스 파일들, 직렬화된 객체들, 이미지들, 도움말 파일들, 그리고 여러 리소스들을 묶는데 사용된다. 이렇게 해서 JAR 파일을 만듦으로 해서 완전한 자바 빈즈 컴포넌트가 생성된다.

4.2 XML을 이용한 자바 빈즈 컴포넌트 명세화

자바 빈즈 컴포넌트의 명세화 실험은 3장 (그림 1)의 컴포넌트 변환-명세화 프로세스 중 4.4, 4.5 단계의 구현을 통해서 확인한다. 구현은 일반적인 자바 어플리케이션에 맞추도록 한다.

4.2.1 명세화 구현 프로그램의 기능적 요구 사항

기능은 크게 4가지로 나눌 수 있으며 다음과 같다.

- 클래스 파일 선택 : 사용자가 명세 정보를 얻기 원하는 자바 빈즈 클래스 파일을 선택하여 입력한다. 빈즈 클래스 파일은 본 논문의 프로세스 과정에 의한 3.4 단계의 산출물이나 제3자 개발에 의한 바이너리 파일 모두 가능하다.
- 클래스로부터 정보 추출 : 입력 받은 자바 빈즈 클래스 파일을 분석하여 정보를 보여준다.
- XML 파일 생성 : 자바 빈즈 클래스 파일을 분석한 정보를 XML 파일로 생성한다.
- 원시 코드 생성 : 분석한 정보를 기반으로 자바 원시 코드를 생성한다.

4.2.2 명세 정보 평가

평가를 위해 입력 예로 사용될 자바 빈즈 컴포넌트를 새로 작성하거나 기존의 컴포넌트를 사용한다. 다른 명세화 도구와 비교/평가하기 위해 JDK에서 제공하는 javadoc이라는 유틸리티를 사용했다. Javadoc 유틸리티는 소스 코드에 정형화된 방식으로 주석을 입력하였을 때 HTML 형식의 문서를 생성하는 프로그램이다. 생성된 HTML 문서는 프레임, 하이퍼 링크, CSS(Cascading Style Sheet)를 지원한다. 프로그래머가 형식에 맞도록 주석 처리를 한다면 상당히 좋은 효과를 거둘 수 있는 도구이나 모든 프로그래머들에게 이러한 것을 기대하기는 어렵다. 게다가 반드시 소스 파일이 있어야만 한다는 단점이 있다.

본 논문에서 제시한 기법을 적용하기 위해서는 컴파일하여 생성된 자바 빈즈 컴포넌트 클래스 파일을 입력한다. 입력을 마친 후, 클래스 파일로부터 정보를 얻기 위해 BeanInfo 객체를 생성하게 된다. 이 객체를 이용하여 자바 빈즈 클래스를 검사하여 정보를 추출하고, XML 문서를 생성한다. 생성된 XML 파일은 (그림 11)과 같다.

XML 파일을 보기 위해 브라우저를 가동시키고 해당 XML 파일을 요청하면 XML, DTD, XSL 파일이 함께 전송되고 브라우저에서 처리되어 XML 문서를 XSL에서 정의된 형식으로 볼 수 있게 된다.

본 논문에서 제시한 기법과의 공정한 비교/평가를 위해 주석 처리는 생략하였다.

다음 <표 4>는 javadoc으로 생성한 HTML 문서와

본 논문에서 구현한 시스템이 생성한 XML 문서의 비교 결과이다.

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="bean.xsl"?>
<!DOCTYPE bean SYSTEM "bean.dtd">
<bean>
<className>Timer</className>
...
<properties><property>
<propertyType>boolean</propertyType>
<propertyName>oneShot</propertyName>
<readMethod>public boolean Timer.isOneShot()</readMethod>
<writeMethod>public void Timer.setOneShot(boolean)</writeMethod>
</property></properties>
```

(그림 11) 생성된 XML 파일의 일부분

〈표 4〉 HTML과 XML의 비교 표

비교 항목	Javadoc으로 생성된 HTML	본 논문에서 생성한 XML
입력 형식	소스 코드	클래스 파일
스타일 쉬트 지원	지원	지원
하이퍼 링크	지원	지원 가능
주석 처리	필요	필요 없음
문서 검증	불가	가능
다중 링크	불가	가능

5. 결 론

소프트웨어 개발 시에 컴포넌트를 사용한다면 여러 가지 효과를 거둘 수 있다. 그러나 컴포넌트화 되지않은 프로그램의 재사용이나 명세화 되지않은 컴포넌트의 재사용 및 유지보수에 많은 어려움이 있다.

본 논문에서는 프로그램의 재사용을 위해서 자바 프로그램에서 자바 빈즈 컴포넌트로 변환시키는 프로세스를 제안하였고 자바 빈즈 컴포넌트로 변환되는 사례를 살펴보았다. 또한 컴포넌트의 명세를 위해서 자바 빈즈 컴포넌트 클래스 파일로부터 명세를 추출하는 방법을 보였다. 자바 빈즈 컴포넌트로부터 명세를 얻어내는 방법은 자바의 내부 검사나 자바 빈즈의 BeanInfo를 이용하는 방법을 통해서 자바 빈즈 컴포넌트의 명세를 추출하였다. 추출된 정보를 사용자에게 적절한 형식의 명세로 표현하는 방법으로는 차세대 웹 언어로 떠오르는 XML을 이용하였다. XML과 더불어 문서의 구조를 정의하는 방법으로는 DTD를 사용하여 정의하였고 이를 사용자에게 적절한 형태로 보이기 위해 스타일 시트 언어인 XSL을 추가하였다. XML을 이용하여 명세를 작성할 때의 장점은 다음과 같다.

1) 자기 서술적으로 명세를 표현 가능
2) 보다 명확하고 구조적이며 문서 자체에서 문서 구조의 검증이 가능
3) 유효하고 잘 구성된 문서 가능
4) 명세 문서의 재사용성이 증대

특히 본 논문에서 제안한 명세의 경우, 클라이언트에서 사용되는 자바 빈즈 컴포넌트, 컴포넌트가 아닌 일반 자바 프로그램의 클래스 파일이 모두 명세가 가능하며, 명세에 필요한 최소한의 정보만을 포함하도록 정의하고 있으므로 간단히 사용할 수 있다는 장점이 있다.

향후, 본 논문에서 제시한 프로세스에 대한 자동화 도구의 구현과 추출된 정보를 기반으로 하는 상세한 자바 원시 코드 생성 부분이 추가될 것이다. 더 나아가 다른 컴포넌트 모델들에 대해서도 XML을 지원하는 명세화 기법이 필요하며 컴포넌트 명세에서 아키텍쳐 명세화로 이루어지는 연구가 필요하다.

참 고 문 헌

[1] Booch, The Unified Modeling Language User Guide, Addison Wesly, 1999.

[2] Bruce Eckel, Thinking in Java, Prentice Hall, 1998.

[3] Christoph Welsch, Alexander Schalk, and Stefan Kramer, "Integrating Forward and Reverse Object-Oriented Software Engineering," Proceedings of the 1997 international conference on Software engineering, 1997, pp.560-561.

[4] Dan Chang, Client/Server Data Access with Java and XML, Wiley, 1998.

[5] Deitel & Deitel, Java How To Program Second Edition, Prentice Hall, 1998.

[6] Graham Hamilton, "JavaBeans Specification Version 1.01, Sun Microsystems, 1997.

[7] Jason Hunter, Java Servlet Programming, O'Reilly, 1998.

[8] Jeffrey Voas, "Maintaining component-based systems," IEEE Software, July/August 1998.

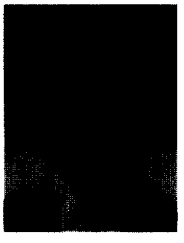
[9] Junichi Suzuki and Yoshikazu Yamamoto, "Making UML Models Interoperable with UXF," <UML> '98 : Beyond the Notation, 1998.

[10] Junichi Suzuki and Yoshikazu Yamamoto, "Managing the software design documents with XML," Proceedings on the sixteenth annual international conference on Computer documentation, 1998, pp.127-136.

[11] Martin Fowler, UML Distilled, Addison Wesly, 1997.

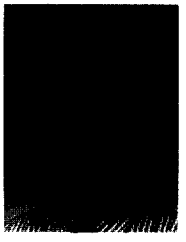
[12] Mary Campione, The Java Tutorial Continued, Ad-

dison Wesly, 1999.
[13] Reaz Hoque, Programming JavaBeans 1.1, McGraw-Hill, 1998.
[14] Richard Light, Presenting XML, Sams, 1998.
[15] Robert C. Seacord, "AGORA : A search engine for software components," IEEE Internet, November 1998.
[16] XML Specification, http : //www.w3c.org/TR/WD-xml.html, 1997.
[17] XSL Specification, http : //www.w3c.org/TR/WD-xsl/, 1998.
[18] 김수동, 실무자를 위한 소프트웨어 공학, 애드텍, 1998.



이 성 은

e-mail : selee@haksan.dsc.ac.kr
1985년 숭실대 전자계산학과 졸업(학사)
1987년 숭실대 대학원 전자계산학과(공학석사)
1994년~현재 숭실대 대학원 박사과정
1987년~1993년 삼성종합기술원 선임연구원
1993년~현재 동서울대학 조교수
관심분야 : 소프트웨어 아키텍처, 소프트웨어 컴포넌트, 소프트웨어 재사용



김 영 익

e-mail : youngick@nownuri.net
1998년 서울산업대 전자계산학과 졸업(학사)
2000년 숭실대 대학원 컴퓨터학과 (공학석사)
2000년~현재 코넷 엔지니어링 개발팀
관심 분야 : 분산 객체, 소프트웨어 컴포넌트

류 성 열

e-mail : syrhew@computer.soongsil.ac.kr
1996년 아주대학교 컴퓨터학부 (공학박사)
1997년~1998년 George Mason University 교환 교수
1981년~현재 숭실대학교 컴퓨터학부 교수
1998년~현재 숭실대학교 정보과학대학원 원장
1998년~현재 숭실대학교 전자계산원 원장
관심분야 : 리엔지니어링, 분산 객체 컴퓨팅, 소프트웨어 재사용