Java 통합 개발 환경에서 기능 컴포넌트들의 상호연동 기법

유 철 중†

요 약

인터넷 및 인트라넷 기반의 클라이언트-서버 프로그램으로 사용 가능한 Java 애플릿 및 애플리케이션의 자동 생성을 위한 통합 개발 환경에서는 메인 윈도, 프로젝트 관리자, 오브젝트 인스펙터, 원시코드 편집기, 폼 디자이너 등과 같은 기능 컴포넌트들간의 상호연동이 필수적이다.

본 논문에서는 인터넷 및 인트라넷 환경에서 모빌 코드 기반의 클라이언트-서버 프로그램 생성 기술인 각각의 기능 컴포넌트들에 대하여 살펴보고, 이들의 상호연동을 위한 자료구조인 'JCode'를 설계 및 구현하며, 이 JCode를 이용한 Two-Way Access 기법의 구조와 동작원리에 대하여 논한다. 이러한 상호연동 기법을 적용함으로써 사용자는 각 기능 컴포넌트들을 사용하여 보다 빠르고 쉽게 Java 프로그램을 작성할 수 있게 된다.

Interconnection Technique of Function Components in the Java Integrated Development Environment

Cheol-Jung Yoo†

ABSTRACT

In the integrated development environments for automatic generation of Java applets and applications that can be used as Internet and Intranet-based client-server programs, it is essential to interconnect between the function components such as main window, project manager, object inspector, source code editor, and form designer.

This paper describes the function components related with the generation of mobile code based client-server program. In then designs and implements a data structure, named 'JCode', for interconnection between the function components. Finally, it discusses the action principles of Two-Way Access technique based on JCode. By applying the proposed JCode based interconnection technique, user can program efficiently using the function components.

1. 서 론

기존의 프로그램 개발 환경이 텍스트 위주에서 윈도 위주로 이전함에 따라 자연스럽게 등장한 비주얼 프로그래밍 개념은 어떤 도구가 어떤 방법으로 얼마만큼 쉬운 프로그래밍 방법을 제공하는가에 기술적 승부를 걸고 있다고 볼 수 있다. 최근의 비주얼 프로그래밍 지원 도구들의 기능별 특징을 간단히 살펴보면 객체지향 프로그래밍 방법 및 컴포넌트 프로그래밍 방법과 다양하고 확장성 있는 컴포넌트를 기본으로 제공해

※ 본 연구는 1997년도 한국전자통신연구원 컴퓨터·소프트웨어기술연구소의 부분적인 지원으로 이루어졌음(위탁과제명 : 모빌코드 기반 클라이언트-서버 프로그램 생성기술 개발).

† 종신회원 : 전북대학교 컴퓨터과학과 교수, 영상·정보신기술 연구소 연구원

논문접수 : 1998년 2월 16일, 심사완료 : 1998년 8월 12일

주고 있으며, 자체 컴포넌트 생성 기능을 지원하여 이를 본 도구에 추가하여 사용할 수 있도록 하고 있다. 또한, 템플리트를 제공하거나 기본 사항 자동 코딩 기능, 실행중 중단 기능, 드래그 앤 드롭 기능, 클릭 앤 드롭 기능과 같이 도구마다 독특한 방식을 지원하고 있으며, 이 외에도 마법사(Wizard)나 전문가(Expert)와 같은 사용자 편의를 위한 자동화 기능도 지원하고 있다.

그러나 WWW와 인터넷에서 플랫폼에 독립적인 언어로서 1991년에 선 마이크로시스템즈(Sun Microsystems)사의 James Gosling외 다수의 개발자에 의해 고안된 Java는 주변 환경 및 다양한 사용자 요구를 충족시키기 위해서 설계와 발전을 계속하고 있는 신생 언어[1, 2]이기 때문에 이를 지원해 줄 수 있는 비주얼 프로그래밍 도구가 현재까지는 안정된 언어 기반의 도구보다 부족한 상태이다. 또한, 인터넷 및 인트라넷 환경에서 클라이언트-서버 프로그램 생성 도구의 기술 개발에 관한 연구가 1990년대 들어와 분산환경에서의 소프트웨어 기술 개발에 관한 관심이 고조되면서 시작되었고, 특히 클라이언트-서버 기반의 분산환경에서 응용들은 주로 브라우저같은 클라이언트 쪽의 디스플레이 기술 개발과 데이터베이스 서버 같은 서버 쪽의 응용 기술 개발로 분리되어 많은 연구를 진행하였다. 그리고 인터넷 및 WWW의 구축 열기가 한창이었던 최근 2~3년 동안은 인터넷과 연동될 수 있는 제반 기술 연구와 기업 인트라넷 응용 기술 개발 방법에 관한 연구에 집중되었다. 특히, 웹 개발 플랫폼이 요구하는 주요 요소 기술은 동적 데이터 출판 기술과 스크립트 지향의 폼(form) 및 리포트(report) 개발과 같은 비주얼 프로그래밍 도구 기술 및 다양한 기능의 컴포넌트 사용과 개발을 지원하는 컴포넌트 프로그래밍 도구 기술 등이 있다.

인터넷 및 인트라넷 기반의 클라이언트-서버 프로그램의 생성을 위한 통합 개발 환경(Integrated Development Environments; IDE)을 플랫폼 독립적이며 인터넷 환경에 적합한 순수 Java(pure Java)로 개발하기 위한 연구는 다양한 기능 컴포넌트(Function Components)의 개발(1단계), 기존의 RDB(Relational Data Base) 및 ODBC(Open Data Base Connectivity)와 연동하기 위한 모듈 개발(2단계), CORBA(Common Object Request Broker Architecture)를 지원하기 위한 모듈 개발(3단계)과 같은 세 가지 단계로 구분하여 볼 수 있다.

본 논문은 위와 같은 요구사항을 만족하기 위한 RAD(Rapid Application Development) 도구의 개발과 관련된 기법에 관한 것으로서, 연구의 주 내용은 메인 윈도(Main Window), 프로젝트 관리자(Project Manager), 오브젝트 인스펙터(Object Inspector), 원시코드 편집기(Source Code Editor), 폼 디자이너(Form Designer) 등과 같은 기능 컴포넌트들에 대하여 기술한 후, 이들 기능 컴포넌트들에서의 변경을 지속적으로 감시하여 각 컴포넌트들의 데이터를 서로 일치시킴으로써 상호연동되도록 하기 위한 자료구조인 JCode를 설계 및 구현하였다. 또한 사용자가 각 기능 컴포넌트들을 사용하여 보다 빠르고 쉽게 Java 프로그램을 생성할 수 있도록 하는 TWA(Two-Way Access) 기법을 개발하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 용어의 정의와 기능 컴포넌트들간의 상호연동 기법에서 이용되고 있는 Java의 객체 감시자인 'Observable' 인터페이스와 'Observer' 클래스의 특징들에 대해서 살펴보고, 3장에서는 인터넷 및 인트라넷 환경에서 모빌(mobile) 코드 기반의 클라이언트-서버 프로그램 생성 기술인 각 기능 컴포넌트들의 구조와 동작 원리에 대해서 논한다. 4장에서는 제안한 자료구조 JCode의 설계 및 구현에 대해서 기술하고 JCode를 기반으로 상호연동을 위한 TWA 기법의 의의를 설명하며, 5장에서는 결론 및 향후 연구방향을 제시한다.

2. 관련 연구

본 논문에서 사용하고 있는 주요 용어에 대한 정의를 간략하게 기술하고, 본 논문의 논점인 기능 컴포넌트들에서 TWA 기법을 구현한 알고리즘의 아이디어가 된 JDK의 'java.lang' 패키지에 있는 'Observable' 인터페이스와 'Observer' 클래스의 객체 감시자에 대한 특징에 대해서 설명한다[2].

2.1 용어 정의

(1) 컴포넌트(components)

- 컴포넌트는 매매할 수 있는 엔티티로서 개방형 시장에서 구입할 수 있는 독립적인 소프트웨어의 이진 형태이다.
- 컴포넌트는 완전한 애플리케이션(application)을 형성하기 위해 다른 컴포넌트와 합해진다.
- 컴포넌트는 예측할 수 없는 결합으로 이용된다.

실세계의 객체와 같이 컴포넌트는 원래 개발자에 의해 전혀 예측되지 않는 방법으로 사용될 수 있다. 전형적으로 컴포넌트는 동일한 패밀리(=suites)의 다른 컴포넌트와 결합된다.

- 컴포넌트는 잘 정의된 인터페이스를 가진다. 객체처럼 컴포넌트는 인터페이스를 통해 다루어진다. 이것은 컴포넌트가 외부에 컴포넌트의 기능을 노출하는 방법이다. 인터페이스를 위해 CORBA/OpenDoc 컴포넌트는 IDL을 제공한다.
- 컴포넌트는 상호운용할 수 있는 객체로서 주소공간, 네트워크, 언어, OS와 무관하게 하나의 객체로 호출될 수 있다.
- 컴포넌트는 확장되는 객체로서 캡슐화, 상속, 다형성을 지원한다는 점에서 진정한 객체이다.

OMG(Object Management Group)에서는 객체지향 데이터베이스, 사용자 인터페이스, 오브젝트 리퀘스트 브로커(Object Request Brocker; ORB) 등 다양한 오브젝트 기술에서 OMG에 준거하는 모델을 코어 오브젝트 모델(Core Object Model; COM)이라고 하고, 각각의 분야가 요구하는 모델을 코어 오브젝트 모델로의 확장으로서 정의될 수 있도록 하고 있다. 이러한 각각의 확장 정의를 컴포넌트라고 부르고 있다[3].

(2) Java Beans

Java를 이용한 프로그램을 재사용 가능하게 한 컴포넌트 소프트웨어로서 비주얼 개발 도구를 이용해 컴포넌트 소프트웨어가 제공하는 속성이나 이벤트를 알아볼 수 있고 속성을 설정해줄 수 있는 설계 시점에서 프로그래밍이 가능한 컴포넌트이다[4, 5].

(3) JDK(Java Development Kit)

Java를 개발한 선 마이크로시스템즈사에서 제공하는 Java 프로그램 개발 도구로서 'bin' 디렉토리에는 실행 파일과 '.dll' 파일이 들어 있으며, 실행 파일은 다음과 같다.

- javac : Java 컴파일러로서 Java 코드를 바이트 코드로 컴파일한다.
- java : Java 인터프리터로서 javac로 컴파일된 바이트 코드를 실행한다.
- jdb : Java 프로그램을 디버깅할 수 있는 명령어 방식의 디버거이다.
- javah : Java 클래스로부터 C 헤더 파일과 C 스터브(stub) 파일을 만들어 주는 도구로서 이를 통해 Java 코드와 C 코드를 연결할 수 있다.
- javap : 컴파일된 Java 파일을 디스어셈블(disassemble)하여 주는 도구로서 바이트 코드를 프린트해 준다.
- javadoc : Java 원시코드로부터 HTML 형태의 API 문서를 만들어 준다.
- appletviewer : 웹 브라우저 없이 애플릿(Applet)을 실행할 수 있게 해주는 도구이다.

(4) JFC(Java Foundation Class)

넷스케이프(Netscape)사의 IFC(Internet Foundation Class)와 JavaSoft의 API를 통합한 JavaBeans형 컴포넌트들의 패키지이다.

2.2 Java의 객체 감시자

(1) 'Observable' 클래스

JDK의 'java.lang' 패키지에 있는 'Observable' 클래스는 프로그램의 다른 부분이 관찰할 수 있는 서브 클래스를 생성하기 위해 <표 1>에서 정의된 메소드를 사용한다. 그러한 서브 클래스의 객체가 변경될 때, 관찰하는 클래스는 통보를 받는다. 관찰하는 클래스는 'update()' 메소드를 정의하고 있는 'Observer' 인터페이스를 구현해야 한다. 'update()'는 관찰된 객체에서 변경을 통보받을 때 호출된다.

〈표 1〉 'Observable'에 의해 정의된 메소드
〈Table 1〉 Method defined by 'Observable'

메 소 드	기 능
void addObserver (Observer obj)	호출 객체를 관찰하는 객체들의 리스트에 obj를 추가한다.
void notifyObservers (Object obj)	update()를 호출함으로써 호출 객체가 변경되었다는 것을 호출 객체의 모든 관찰자에게 통보한다. obj는 update()에 대한 인자로서 전달된다.
void setChanged()	호출 객체가 변경되었을 때 호출된다.

(2) 'Observer' 인터페이스

관찰할 수 있는 객체를 관찰하기 위해 'Observer' 인터페이스를 구현해야 한다. 이 인터페이스는 다음과 같은 하나의 메소드만을 정의한다.

void update(Observable observOb, Object arg)

여기에서 '*observOb*'는 관찰되고 있는 객체이고, '*arg*'는 'notifyObservers()'에 의해 전달된 값이다. '*update()*' 메소드는 관찰된 객체에서 변경이 일어날 때 호출된다.

3. 기능 컴포넌트들의 구조 및 동작 원리

본 논문에서는 Java 애플릿 및 애플리케이션 작성을 위한 IDE에서 단일 목적을 위해서 수행되는 메인 윈도(Main Window), 프로젝트 관리자(Project Manager), 오브젝트 인스펙터(Object Inspector), 원시코드 편집기(Source Code Editor), 폼 디자이너(Form Designer) 등을 **기능 컴포넌트**들(Function Components)로 정의하였다.

이러한 기능 컴포넌트들은 시스템 및 운영체제 독립적인 도구 환경의 구축과 이식작업의 용이성, 그리고 Java Beans 및 RMI(Remote Method Invocation)와 같은 분산 객체 기술을 지원하고 있는 Java 언어의 이점[4]을 활용하기 위하여 Java로 구현하였다. Java는 열거한 이점 외에도 <표 2>에서 보는 것처럼 다른 언어에 비하여 개발자의 다양한 요구사항을 충족시킬 수 있는 풍부한 특성을 가지고 있다[6].

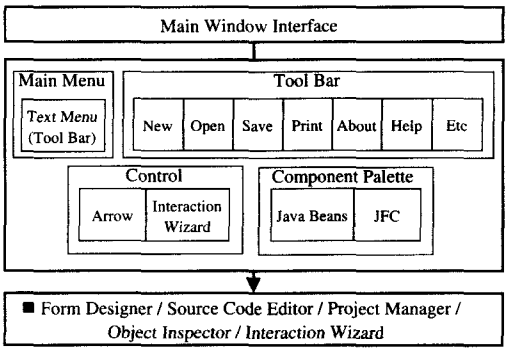
〈표 2〉 Java 언어와 다른 언어와의 특성 비교
〈Table 2〉 Comparison of Java and other languages

언어 / 특성	Java	SmallTalk	TCL	Perl	Shells	C++	C
간결성	●	●	●	◑	◑	○	◑
객체지향	●	●	○	●	○	◑	○
견고함	●	●	●	●	●	○	○
보안성	●	◑	◑	●	◑	○	○
인터프리터	●	●	●	●	●	○	○
동적	●	●	●	●	◑	○	○
이식성	●	◑	●	●	◑	◑	◑
구조중립	●	◑	◑	●	◑	○	○
쓰레드	●	○	○	●	○	○	○
예외	●	●	○	●	○	◑	○
수행 능력	높음	중간	낮음	중간	낮음	높음	높음

● : 있음 ◑ : 다소 있음 ○ : 없음

3.1 메인 윈도

메인 윈도는 본 도구의 전체적 기능을 제어하며 사용자의 세부적 요구사항을 처리하기 위한 기능 컴포넌트로서 (그림 1)과 같은 구조로 되어 있다. 메인 윈도에는 메뉴와 툴바, 컴포넌트 팔레트가 있으며, 특히 컴포넌트 팔레트에는 Java Beans의 아이콘이 종류별로 각각의 페이지에 정리되어 있다. 또한, JDK 기본 패키지에 있던 기존의 컨테이너나 AWT(Abstract Window Toolkit) 및 대화상자도 모두 Java Beans로 되어 있기 때문에[5] 이들 컨트롤을 프로젝트 관리자나 폼 디자이너 추가하면 자동으로 그에 대한 패키지가 편입(import)되고 원시코드가 만들어진다.



(그림 1) 메인 윈도의 구조
(Fig. 1) Architecture of main window

3.2 폼 디자이너

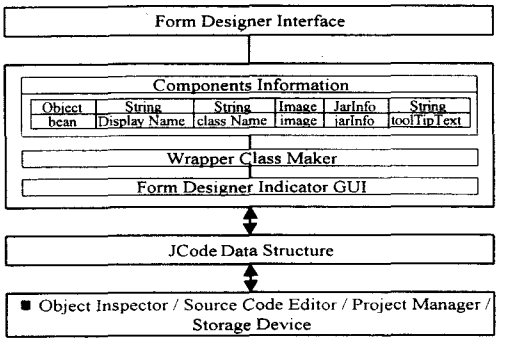
폼 디자이너는 사용자가 메인 윈도의 컴포넌트 팔레트에서 'Button', 'TextField', 'TextArea', 'RadioButton', 'Scrollbar' 등과 같이 다양한 기능을 가진 컴포넌트들을 마우스로 '클릭 & 드롭'하여 사용자 프로그램의 GUI 인터페이스를 작성하는 바탕이 되는 폼(form)으로서 Java 애플릿 및 애플리케이션의 작성을 시각화(visualize)하여 준다[7, 8 ,9].

(그림 2)와 같은 구조로 이루어진 폼 디자이너는 사용자가 폼을 가진 파일을 열 때 동적으로 생성되는 것이 기본 원리이며 각각의 컴포넌트 및 프레임의 저장은 JCode의 자료구조를 이용한 객체 저장 방법을 사용하고 있다[10].

3.3 프로젝트 관리자

사용자가 작성하려는 Java 프로그램의 프로젝트를 효과적으로 조직화하여 관리하기 위한 기능 컴포넌트인 프로젝트 관리자는 중첩된 라이브러리나 폼 디자이너들간의 의존관계를 제어한다. 본 도구에서 새로운 프로젝트를 생성하거나 기존의 프로젝트를 로드할 때, 프로젝트 관리자는 Java 원시코드를 자동적으로 분석하는 백그라운드 원시코드 분석기(background source code parser)를 작동하고, JCode를 이용하여 프로젝트에 대한 정보 뱅크를 만든다. 이 정보는 원시코드 편집기에서 프로젝트 안의 모든 Java 클래스들에 적용되

는 빠른 검색에 이용되며 프로젝트 관리자와 연동되어 동작하는 그 밖의 기능 컴포넌트들의 제어에 사용된다.



(그림 2) 폼 디자이너의 구조
(Fig. 2) Architecture of form designer

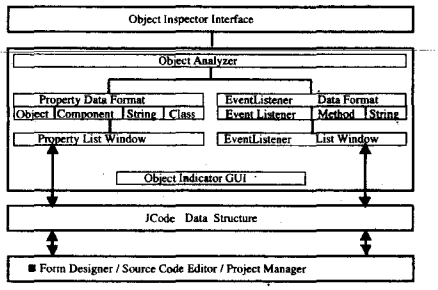
3.4 오브젝트 인스펙터

폼 디자이너에서 작성된 컴포넌트들의 정보를 다루기 위한 오브젝트 인스펙터는 RAD 계열의 도구에서 없어서는 안될 중요한 기능 컴포넌트들 중의 하나이며, 폼 디자이너 및 원시코드 편집기와 상호연동하여 동작한다. 사용자가 JDK 1.1 버전 이상의 AWT에서 제공되는 컴포넌트(라벨, 버튼, 텍스트 상자, 스크롤 바, 선택 버튼 등)들과 제3의 업체(third party)에서 Java Beans형으로 제공되는 다양한 종류의 컴포넌트들을 폼 디자이너에 삽입하여 Java 프로그램의 그래픽 사용자 인터페이스(GUI)를 설계할 때, 오브젝트 인스펙터는 폼 디자이너에서 선택된 컴포넌트의 속성과 이벤트 리스너들의 정보를 'Property List'와 'EventListener List' 윈도에 보여주며 사용자로 하여금 이들 정보를 보다 편리하게 변경할 수 있도록 하여준다.

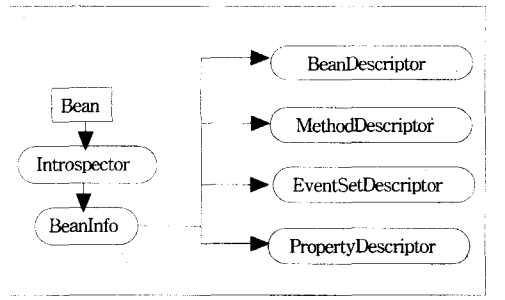
오브젝트 인스펙터의 구조는 (그림 3)과 같으며 폼 디자이너에 삽입된 컴포넌트의 속성, 이벤트 리스너, 메소드와 같은 정보를 JDK의 'java.beans' 패키지를 이용하여 추출한 후 그 정보를 GUI를 통하여 사용자에게 시각화하여 주고 변경시킬 수 있게 함으로써 프로그램 개발을 용이하게 한다[5].

(1) Property List Window

오브젝트 인스펙터의 'Property List'는 폼 디자이너에 삽입된 빈(bean)의 속성 정보를 (그림 4)와 같은 구



(그림 3) 오브젝트 인스펙터의 구조
(Fig. 3) Architecture of object inspector



(그림 4) 'Bean' 클래스의 정보 추출 과정
(Fig. 4) Information extraction process of 'Bean' class

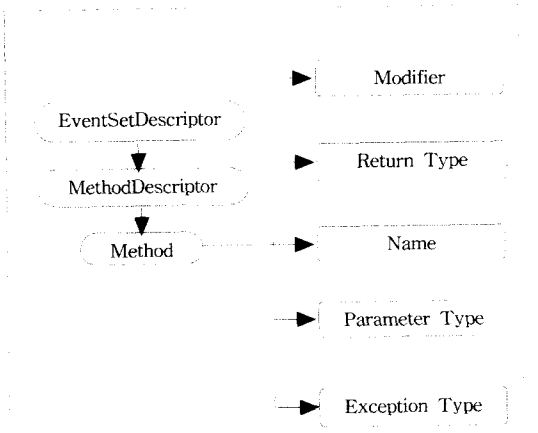
조로 되어 있는 'Bean' 클래스의 정보추출 과정을 통해서 'PropertyDescriptor' 클래스를 이용하여 속성의 이름, 자료형, 'getter', 'setter' 메소드를 얻는다[5, 11].

```
BeanInfo bi = Introspector.getBeanInfo(bean.getClass());
PropertyDescriptor properties[];
properties = bi.getPropertyDescriptors();
```

위의 경우는 오브젝트 인스펙터에서 'PropertyDescriptor'를 추출하는 과정을 구현한 예로서, 'BeanInfo' 클래스형으로 선언된 'bi'는 'Introspector' 클래스의 'getBeanInfo()' 메소드를 호출함으로써 빈에 관한 정보를 얻게되며, 'PropertyDescriptor' 클래스형 배열로 선언된 properties는 'BeanInfo' 클래스의 'getPropertyDescriptors()' 메소드를 호출함으로써 속성정보를 얻는다. 그리고 'Property List'에서 사용자가 값(value)을 변경시키면 자동으로 빈의 성질이나 형태가 변경되는데, 이는 'PropertyDescriptor'와 'MethodDescriptor' 클래스를 사용하여 관리하도록 하였다.

(2) EventListener List Window

오브젝트 인스펙터의 'EventListener List'는 컴포넌트 또는 빈이 발생시키는 이벤트에 대응하는 'EventListener'의 메소드 리스트를 제공함으로써 사용자는 이 이벤트 메소드의 본체만 정의해 주면 된다[5, 11]. 이러한 작업은 (그림 5)와 같은 구조로 되어 있는 컴포넌트와 빈의 정보 추출과정 중 'EventSetDescriptor' 클래스를 이용하여 'EventListener'의 메소드 이름(name), 반환 형(return type), 변경자(modifier), 매개변수 형(parameter type), 예외상황 형(exception type)과 같은 정보를 얻는다.



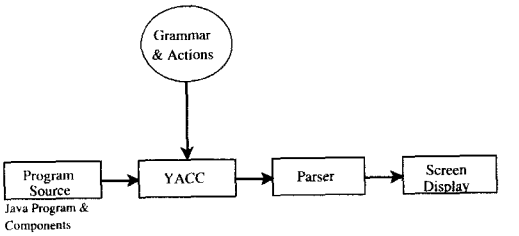
(그림 5) 'EventListener' 메소드의 정보 추출 과정
(Fig. 5) Information extraction process of 'EventListener' method

```
BeanInfo bi = Introspector.getBeanInfo(bean.getClass());
EventSetDescriptor eventSets[];
eventSets = bi.getEventSetDescriptors();
MethodDescriptor methods[];
methods = eventSets[0].getListenerMethodDescriptors();
Method method = methods[0].getMethod();
```

위의 경우, 폼 디자이너상에 삽입된 빈에 대한 'EventListener'의 정보를 추출하기 위하여 'BeanInfo' 클래스형으로 선언된 'bi'는 'Introspector' 클래스의 'getBeanInfo()' 메소드를 호출함으로써 컴포넌트와 빈에 관한 정보를 얻게 되며 'EventSetDescriptor' 형 배열로 선언된 'eventSets'은 'BeanInfo' 클래스의 'getEventSetDescriptors()' 메소드를 호출함으로써 (그림 5)와 같은 'EventListener' 메소드의 정보 추출 과정을 통하여 'MethodDescriptor' 클래스 형 배열인 'methods'에 'EventListener' 메소드의 정보 값(value)을 저장한다.

3.5 원시코드 편집기

원시코드 편집기는 사용자가 폼 디자이너에서 작성하고자 하는 프로그램의 GUI를 설계한 후 원시코드 편집기에서 직접 본체 코드를 입력하거나 수정할 수 있게 하여주는 기능 컴포넌트이다. 원시코드 편집기의 구조는 (그림 6)과 같으며 다양한 색상으로 표시되는 문장 분류 체계를 지원하기 위하여 Java 문법 및 패키지를 분석하는 파서(parser)를 내장하고 있고 Java 프로그램 및 표준 클래스 라이브러리에서 Grep과 같은 다양한 검색 기능을 지원하고 있다.



(그림 6) 원시 코드 편집기의 구조
(Fig. 6) Architecture of source code editor

4. JCode의 설계 및 구현

4.1 개 념

3장의 기능 컴포넌트들의 구조 및 작동원리에서 보았듯이, Java 프로그램의 작성을 위하여 사용자가 폼 디자이너에서 Java 개발 도구인 JDK의 AWT 패키지 안에 있는 기본 컴포넌트들과 JFC의 컴포넌트들을 임의의 폼 디자이너에 배치시킬 때마다 각각의 컴포넌트들에 대한 정보를 저장하는 것이 필요한데, 본 장에서는 이를 위한 클래스형 자료구조인 JCode를 설계하고 구현하였다. 그리고 Java 애플릿 및 애플리케이션 생성을 위한 IDE에서, 사용자가 기능 컴포넌트들을 효율적으로 제어하여 빠르고 편리하게 Java 프로그램을 개발하기 위한 목적으로 TWA 기법을 개발하였다. 이를 지원하기 위하여 제안된 JCode는 추상 데이터 타입, 상속, 다형성, 캡슐화와 같은 객체지향 프로그래밍 언

어의 특징을 가지고 있는 자료구조로서, 각각의 단일 목적을 가진 기능 컴포넌트들의 상호연동이 가능하도록 해준다. 본 논문에서 제안한 JCode 및 이와 관련한 TWA 기법을 논하기 위하여 다음과 같은 몇 가지 정의가 필요하다.

다음은 기능 컴포넌트들에 대한 정의이다.

[정의 1] *기능 컴포넌트*

하나의 프로젝트 P에 존재할 수 있는 기능 컴포넌트들을 각각 다음과 같이 정의한다.

I : 오브젝트 인스펙터

M : 프로젝트 관리자

F : 폼 디자이너의 집합

즉, $F = \{f_i \mid f_i$는 임의의 *Form Designer*, $i = 1, 2, \ldots, n\}$

S : 원시코드 편집기의 집합

즉, $S = \{s_i \mid s_i$는 임의의 *Source Code Editor*, $i = 1, 2, \ldots, o\}$

제안될 JCode는 하나의 프로젝트 내에 다음과 같은 집합으로서 존재한다.

[정의 2] *JCode의 집합*

하나의 프로젝트 P에 존재할 수 있는 일련의 JCode의 집합을 J라 한다. 따라서 J는 다음과 같다.

$J = \{j_i \mid j_i$는 임의의 *JCode*, $i = 1, 2, \ldots, l\}$

따라서 하나의 프로젝트는 다음과 같이 정의될 수 있다.

[정의 3] *프로젝트 P*

하나의 프로젝트 P는 정의 1과 2에 의하여 다음과 같이 정의된다.

$$P = (F, S, I, J, M)$$

정의된 임의의 JCode와 관련하여 빈 노드(bean node)들의 집합은 다음과 같이 정의된다.

[정의 4] *빈 노드의 집합*

임의의 JCode j_i에서 생성되는 빈 노드들의 집합을 B라 한다. 따라서 B는 다음과 같다.

$B = \{b_i \mid b_i$는 임의의 빈 노드, $i = 1, 2, \ldots, m\}$

하나의 프로젝트에 대한 임의의 관점은 다음과 같이 정의될 수 있다.

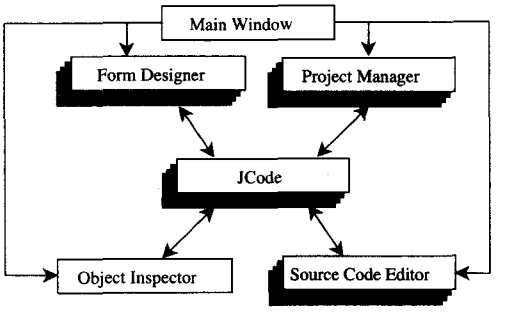
[정의 5] *프로젝트 P에 대한 임의의 관점*

B와 관계 있는 J의 임의의 원소인 JCode j_i를 후보자(Candidate) $C_B^{j_i}$라 할 때, 이와 관련된 프로젝트 P에 대한 임의의 관점 $View_i(P)$는 다음과 같이 정의된다.

$$View_i(P) = (f_i, s_i, I, C_B^{j_i}, M)$$

여기에서 f_i와 s_i는 각각 $f_i \subset F$, $s_i \subset S$ 이며 IDE에서 JCode를 생성할 때 사용된다.

(그림 7)에서 보듯이 각 기능 컴포넌트들은 JCode를 통해 항상 변경을 서로 감시하여 컴포넌트들의 데이터가 구성요소간에 서로 일치되도록 상호연동된다. 예를 들어, 오브젝트 인스펙터에서 어느 컴포넌트의 속성값을 변경하면 오브젝트 인스펙터는 이 변경을 JCode에 저장하고, 원시코드 편집기와 폼 디자이너 및 프로젝트 관리자는 항상 JCode를 감시하여 변경이 감지되면 곧바로 JCode의 내용을 각각 반영한다.



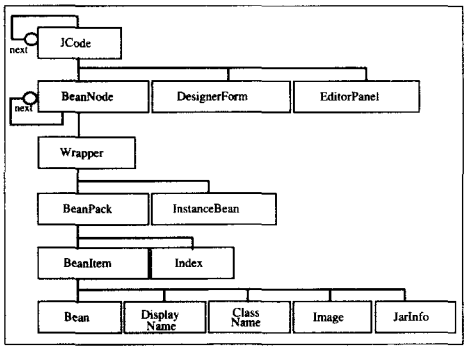
(그림 7) 기능 컴포넌트들간의 자료 전달
(Fig. 7) Data transfer between function components

다음 절에서는 자료구조 JCode와 각각의 기능 컴포넌트들이 JCode를 기반으로 실행되는 방법에 대해서 논하고, JDK의 'java.lang'에서 제공되는 'Observable' 인터페이스와 'Observer' 클래스의 특징을 이용한 TWA 기법의 동작원리에 대해서 설명한다.

4.2 JCode의 자료구조

JCode는 초기 생성시 임의의 메모리에 동적 할당을 위하여 벡터(vector)를 이용한 선형 리스트 구조로서 (그림 8)과 같이 구성되어 있으며, (그림 9)는 'JCode' 클래스의 구현 코드를 보여주고 있다. 'JCode' 클래스의 멤버인 'BeanNode' 클래스는 폼 디자이너가 임의의 컴포넌트들을 포함할 수 있기 때문에 이러한 컴포넌트

들에 대한 정보를 저장하기 위한 클래스로서, 메모리에서 동적으로 할당된 JCode내의 정적 비선형 트리 구조이며 (그림 10)과 같이 정의되어 있다. 또한, 'JCode' 클래스는 사용자가 작성한 폼 디자이너와 원시코드 편집기에 대한 식별자를 저장하기 위하여 (그림 8)에서 보는 바와 같이 'DesignerForm' 클래스와 'EditorPanel' 클래스를 멤버 변수로 가지고 있다. 비선형 트리 구조인 'BeanNode' 클래스는 폼 디자이너에서 컴포넌트를 다루기 위한 'Wrapper' 클래스를 멤버 변수로 가지고 있으며, 이 'Wrapper' 클래스는 사용자가 선택한 컴포넌트를 인스턴스화 하기 위하여 Java의 'Object' 형으로 선언된 'InstanceBean'과 JDK의 기본 AWT 컴포넌트 및 JFC의 컴포넌트들을 다루기 위한 'BeanPack' 클래스를 멤버 변수로 가지고 있다.



(그림 8) 'JCode' 클래스의 구조
(Fig. 8) Structure of 'JCode' class

```
class JCode extends Observable
{
   // --- Private ---
   private int formType;
   private DesignerForm designerForm;
   private EditorPanel editorPanel;
   private BeanNode root;
   // --- Public ---
   public int getFormType()
   public void setDesignerForm(DesignerForm form)
   public DesignerForm getDesignerForm()
   public void setEditorPanel(EditorPanel panel)
   public EditorPanel getEditorPanel()
}
```

(그림 9) 'JCode' 클래스
(Fig. 9) 'JCode' class

```
public class BeanNode implements ComponentListener
{
   private JCode      jcode;
   private Container container;
   private String     beanName;
   private Vector     childs = new Vector();

   public BeanSet(JCode jcode, Container container)
   public Container getContainer()
   public Object getBean()
   public Object getInitBean()
   public void setBeanName(String beanName)
   public String getBeanName()
   public String getClassName()
   public int getBeanIndex()
   public BeanPack getBeanPack()
}
```

(그림 10) 'BeanNode' 클래스
(Fig. 10) 'BeanNode' class

본 논문의 IDE에서는 세가지 종류의 컴포넌트를 지원하고 있는데, 그 종류는 다음과 같다.

① JDK의 'java.awt' 패키지에 있는 컴포넌트들
② JFC의 'com.sun.java.swing' 패키지에 있는 컴포넌트들
③ BDK를 이용한 사용자 컴포넌트들

①과 ②의 컴포넌트들은 사용자를 위해서 IDE내에서 기본적으로 제공되도록 구현하였으며, ③은 사용자가 정의한 컴포넌트로서 메인 윈도의 컴포넌트 팔레트에 추가하여 사용할 수 있다. 그런데 IDE에서 ①과 ②의 컴포넌트들은 JDK와 JFC에서 '.class' 파일로 제공되고 있기 때문에 내부적으로 어떠한 변환 처리를 할 필요없이 사용할 수 있지만, ③은 BDK로 작성될 때 '.jar' 파일로 압축되어 있어서 사용자가 폼 디자이너에 '클릭 & 드롭'할 경우 반드시 '.class' 파일로 변환 처리되어야 하며, 이 변환 처리는 BDK의 'JarInfo' 클래스를 이용하고 있다.

BDK는 Java Beans를 생성하기 위한 컴포넌트 개발 도구로서 BDK에 의하여 생성된 컴포넌트는 '.jar' 파일로 압축되는데, 본 논문의 IDE에서는 이러한 사용자 컴포넌트도 메인 윈도의 컴포넌트 팔레트에 추가시켜 사용할 수 있도록 하고 있다. IDE에서 컴포넌트를 메모리로 인스턴스화 하기 위해서는 사용자 컴포넌트가 저장 매체인 하드디스크에 '.jar' 파일로 압축되어 있기 때문에, 먼저 JDK의 'java.io' 패키지에 있는 'ZipInputStream' 클래스를 이용하여 Zip 압축을 해제한 후, JDK의 'java.lang' 패키지에 있는 'ClassLoader'를 이용하여 '.jar' 파일의 정보를 'JarInfo' 클래스에 저장

하는 두 단계로 수행된다.

'BeanPack' 클래스는 파일 형식이 다른 위의 ①과 ②의 컴포넌트와 ③의 컴포넌트들에 대한 데이터와 메소드를 객체지향 패러다임의 주요 특징인 다형성(polymorphism)과 캡슐화(encapsulation) 개념을 사용하여 구현하였다. 'BeanPack'의 'Index' 변수는 저장된 컴포넌트 테이블에서 ①과 ②를 식별하기 위한 인덱스로 사용되고 있으며, 'BeanItem' 클래스는 컴포넌트가 인스턴스화 될 때 객체를 저장하기 위한 Java의 'Object' 형 'Bean' 변수, 제거 화면상에 표시되는 툴팁(tooltip)의 내용을 저장하기 위한 'String' 형 'DisplayName' 변수, 컴포넌트의 패키지와 이름을 저장하기 위한 'String' 형 'ClassName' 변수, 컴포넌트의 아이콘을 저장하기 위한 'Image' 형 변수, 그리고 ③번 컴포넌트를 다루기 위한 'JarInfo' 형 변수를 멤버 변수로 가지고 있다.

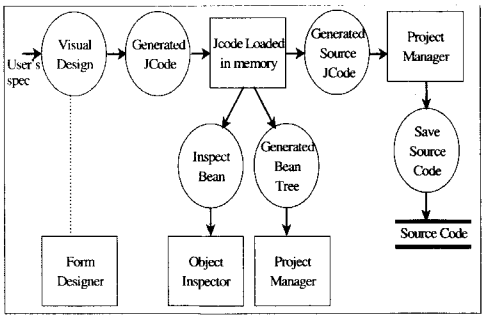
4.3 JCode와 기능 컴포넌트들의 상호연동

제안된 JCode와 기능 컴포넌트들간의 상호연동은 (그림 11)에서 보는 바와 같이 다음과 같은 일련의 동작들이 연속적으로 수행됨으로써 이루어진다.

① 사용자가 요구명세에 의해 폼 디자이너상에서 컴포넌트를 이용한 비주얼 디자인을 하면 자동적으로 메모리에 적재된 JCode의 자료구조에 사용자의 비주얼 디자인 속성 명세가 저장된다.

② 생성된 JCode를 바탕으로 원시코드 생성기를 통해서 Java 원시코드를 생성하며 생성된 원시코드는 파일단위로 하드디스크에 저장된다.

③ JCode에 저장된 컴포넌트들을 조사하여 해당 정보를 오브젝트 인스펙터에 전달함으로써 오브젝트 인스펙터의 기능을 실행할 수 있게 한다.



(그림 11) 설계된 IDE에서의 초기화 과정
(Fig. 11) Initialization process in the IDE designed

④ 사용자는 여러 개의 폼 디자이너를 호출하여 각각의 컴포넌트들을 디자인할 수 있기 때문에, JCode는 재귀적 생성 관계를 가지고 있으며 디자인된 컴포넌트들을 트리 구조로 저장하여 프로젝트 관리자에 그 정보를 넘겨준다.

이와 같이 사용자가 IDE에서 컴포넌트들을 이용하여 폼 디자이너상에 Java 프로그램의 GUI를 디자인하면 JCode를 메모리에 생성한다. 비주얼 프로그래밍 방식과 컴포넌트 프로그래밍 방식을 지원하는 개발 도구들은 기존의 텍스트 프로그래밍 개발 도구에서 원시코드를 직접 타이핑함으로써 발생하는 프로그램 생산성 저하를 개선하는 데 많은 기여를 하고 있다. 여기에서의 JCode는 사용자 요구사항 명세를 만족하는 정보를 가지고 있으면서 사용자가 프로그램의 원시코드를 직접 타이핑하지 않고도 원하는 프로그램을 손쉽고 빠르게 개발할 수 있도록 하여주는 핵심 자료구조라고 할 수 있다.

다음은 한 프로젝트 내에 존재하는 기능 컴포넌트들간의 관련성을 카디널리티(cardinality) 측면에서 정의한 것이다.

[정의 6] *기능 컴포넌트들간의* ***카디널리티 관련성***

기능 컴포넌트들이 공존하는 IDE의 한 프로젝트 P에서 사용자가 작성한 폼 디자이너의 집합 F가 존재할 때, F와 JCode의 집합 J와의 관계는 다음과 같이 정의된다.

$$n(F) = n(J)$$

또한, F와 원시코드 편집기의 집합 S와의 관계는 다음과 같이 정의된다.

$$n(F) \leq n(S)$$

여기에서 $n(F)$, $n(J)$, $n(S)$는 각각 집합 F, J, S의 카디널리티이다.

4.4 Two-Way Access 기법

기존의 프로그램 개발 방법인 텍스트 위주의 방식에서는 사용자가 직접 원시코드를 입력하였는데, 윈도 기반의 RAD 도구나 IDE에서의 비주얼 프로그래밍과 컴포넌트 프로그래밍 개발 방식은 시각적이고 기능적인 방법을 이용하고 있다. 이러한 개발 방식에서 사용하는 대표적인 TWA 기법은 프로그램 개발시 원시코드 편집기에서 텍스트를 직접 입력하는 방식뿐만 아니

라 폼 디자이너에서 다양한 종류의 컴포넌트들을 사용하여 프로그램을 개발할 수 있도록 지원하는 기술이다. 본 논문에서 설계·구현된 기능 컴포넌트들 중 TWA 기법을 지원하는 것은 폼 디자이너와 원시코드 편집기이다. 즉, 사용자가 폼 디자이너 윈도에서 컴포넌트를 이용하여 Java 프로그램을 작성하다가 원시코드 편집기 윈도로 포커스(focus)를 이동시켜 텍스트 타이핑 방법에 의한 작성을 계속해서 할 수 있으며 그 반대의 경우도 가능하게 되는 것이다. 컴포넌트의 속성과 이벤트를 보다 효과적으로 제어하기 위한 오브젝트 인스펙터와 하나의 프로그램내에서 중첩된 폼 디자이너의 의존관계를 조직적으로 제어하기 위한 프로젝트 관리자는 JCode에 기반하여 (그림 12)와 같이 상호연동되어 동작한다.

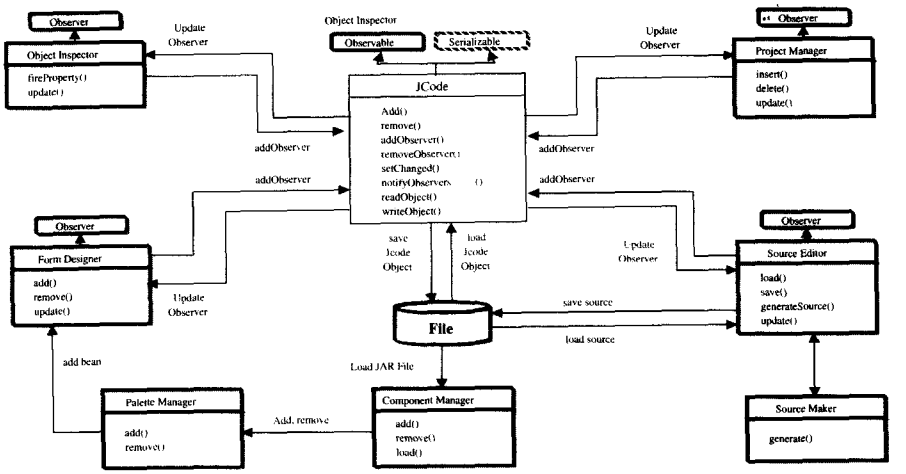
여기에서 임의의 컴포넌트와 관련된 성질에 대한 정의는 다음과 같다.

[정의 7] *컴포넌트의 성질*

[정의 1]로부터 임의의 폼 디자이너 f_i는 $f_i \subset F$이다. 여기에서 임의의 컴포넌트를 x라 할 때, x가 f_i에서 사용될 수 있다면 이 x는 JDK의 컴포넌트이거나 JFC의 컴포넌트 또는 사용자 컴포넌트 중 하나이다(4.2절 참조).

(그림 12)로부터 TWA 기법을 자세히 설명하면 다음과 같다. JCode는 (그림 9)에서 보는 바와 같이 JDK의 'java.lang' 패키지에 있는 'Observable' 클래스로부터 상속을 받고 있고, 각각의 기능 컴포넌트들(폼 디자이너, 원시코드 편집기, 프로젝트 관리자, 오브젝트 인스펙터)은 'Observer' 인터페이스로 구현된 Java 클래스내에서 'addObserver(Object O)' 메소드를 이용하여 객체 감시자가 되며 Java 가상 머신(Java Virtual Machine; JVM)으로부터 각각 'update(Observable O, Object arg)'를 통해 JCode의 변경에 대한 값을 반환받기 때문에, JCode는 JVM에서 감시 대상이 되는 객체가 되는 동시에 각각의 기능 컴포넌트들은 JCode를 감시하는 감시자 역할의 객체가 된다. 즉, 감시자들인 각각의 기능 컴포넌트들 중 임의의 한 기능 컴포넌트가 JCode의 내용을 변경시키면 'Observable' 메소드는 메모리에 생성된 JCode 클래스의 내용이 임의의 한 기능 컴포넌트에 의해 변경될 때마다 자동으로 변경된 정보가 있음을 각각의 기능 컴포넌트에 알린다. 이때, 일단 한번 JCode가 사용자의 디자인 요구명세에 따라 메모리에 생성되면 IDE에서 비주얼 프로그래밍과 컴포넌트 프로그래밍 방법만으로도 Java 애플릿 및 애플리케이션의 개발이 가능하다.

TWA 기법에 기반한 주 작업은 폼 디자이너와 원시코드 편집기에서 수행되며 오브젝트 인스펙터와 프로젝트 관리자 및 인터액션 마법사는 TWA 기법을 보조하는 데 사용된다.

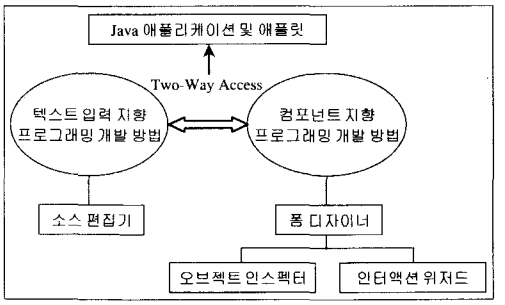


(그림 12) TWA 기법의 개념도
(Fig.12) Conceptual diagram of TWA technique

4.5 제안 기법의 의의

4세대 언어는 결과를 성취하는데 필요한 행위들을 기술하기보다는 원하는 결과들을 기술해 주는 프로그램 언어를 말한다[12]. 본 논문은 4세대 객체지향 언어인 'Java'를 지원하면서 프로그램을 빠르게 개발하기 위한 4세대 기술(4GT: 4th Generation Techniques)에 대한 것으로서 Java 언어가 가지고 있는 보안 및 이식성의 장점을 살리기 위하여 순수 Java 언어를 사용하여 객체지향 설계 및 개발을 하고 있다. 이러한 4세대 기술에 기반을 둔 RAD 도구상의 다양한 종류의 기능 컴포넌트들은 3장에서 설명한 것처럼 4세대 언어의 특징을 사용자에게 지원할 수 있도록 상호운용적인 활동(activities) 및 책임(responsibilities)을 가지고 있다.

본 논문에서는 Java를 지원하는 RAD 도구 환경에서 기능 컴포넌트들 사이의 상호연동을 지원하기 위하여 이들간의 공통 자료구조인 'JCode'를 설계 및 구현하였으며, 이를 기반으로 한 TWA 기법의 원리를 소개하였다. 제안된 TWA 기법은 (그림 13)과 같이 사용자가 RAD 도구상에서 Java 애플리케이션 및 애플릿을 작성할 때 기존의 텍스트 입력 지향 방법과 컴포넌트 지향 방법을 모두 지원하기 위한 기술이다.



(그림 13) TWA의 사용 사례
(Fig. 13) Use case of TWA

TWA 기법은 다음과 같은 특징을 포함하고 있다.

첫째, 제안된 TWA 기법은 IDE에서 사용자가 Java 프로그램을 개발하는데 있어서 원시코드 편집기에서 코드를 직접 입력하는 텍스트 지향 프로그래밍 방식뿐만 아니라 폼 디자이너, 오브젝트 인스펙터, 프로젝트 관리자 등과 같은 기능 컴포넌트들을 이용하는 컴포넌트 지향 프로그래밍 방식을 더불어 지원함으로써 최근의 RAD 도구들의 장점을 구현하는 것이 가능하다.

둘째, 본 도구의 임의의 기능 컴포넌트들에서 선택된 컴포넌트는 JCode에 의해서 객체 단위로 다루어진다. 따라서 기능 컴포넌트들 사이에서 상속과 다형성 및 동적 바인딩을 지원한다.

셋째, 본 도구는 순수 Java를 이용하여 구현하였기 때문에 선택된 컴포넌트는 기능 컴포넌트들 사이에서 상속, 다형성, 동적 바인딩과 같은 Java의 객체지향 특징을 지원한다.

넷째, 본 도구에서 기능 컴포넌트들간의 상호연동을 가능하게 하는 JCode는 사용자로 하여금 프로그램을 보다 빠르게 개발할 수 있는 기반 기술이다. 특히, JCode는 Java가 갖는 특성의 기반 기술을 활용하였기 때문에 Java의 발전과 함께 향후 개발될 여러 RAD 도구들의 기반 기술이 될 수 있다.

TWA 기법에서 기능 컴포넌트들간의 변경 사항 감지 및 처리를 구현하기 위하여 이용하고 있는 JDK의 'Observable/Observer'는 기능 컴포넌트들의 객체 상태 변화를 알아내기 위한 용도로 사용하였다. 본 논문의 기능 컴포넌트들은 RAD 환경 지원을 위하여 각각 고유의 기능 및 자료구조를 가지고 있기 때문에 서로 상이한 기능에 대해서 자료 변환 처리를 능동적으로 대처할 수 있다. 이러한 능동적인 자료 변환 처리 능력이 'JCode'의 핵심 이론이며 RAD 도구에서 객체지향 패러다임을 지원가능하게 하는 기술이라 할 수 있다. 이러한 면에서 볼 때 본 논문은 RAD 도구에서 기능 컴포넌트들간의 상호운용성 지원을 위한 공통 자료 저장소 'JCode'를 제시하였다는 데 그 의의가 있다.

5. 결론 및 향후 연구과제

본 논문은 인터넷 및 인트라넷 환경에서 모빌(mobile) 코드 기반의 클라이언트-서버 프로그램 생성 기술 중 Java 애플릿 및 애플리케이션의 개발을 용이하게 하여 주는 RAD 도구를 플랫폼 독립적인 Java 언어로 구현하는 연구의 일부로서, 메인 윈도, 프로젝트 관리자, 폼 디자이너, 원시코드 편집기, 오브젝트 인스펙터 등과 같은 각각의 기능 컴포넌트들이 상호연동하여 동작할 수 있도록 연동을 위한 자료구조인 JCode를 설계하고 구현하였다. 또한 임의의 Java 프로그램을 개발하는데 있어서 단일 목적을 가진 기능 컴포넌트들간의 상호연동을 위해 TWA 기법을 적용하였다.

먼저 구현된 기능 컴포넌트들의 구조와 동작 원리에 대해 살펴본 후 이들의 상호연동을 위한 자료구조

로서 설계 및 구현된 JCode에 대하여 논하였다. 그리고 JCode를 기반으로 기능 컴포넌트들간의 상호연동을 가능하게 하는 TWA 기법에 대하여 기술하였다. 기능 컴포넌트들간의 상호연동을 가능하게 하는 TWA 기법을 지원하는 자료구조인 JCode는 본 도구에서 사용자로 하여금 보다 빠르고 편리한 Java 프로그램을 작성할 수 있는 기반이 될 것이다. 또한 IDE에서 사용자가 Java 프로그램을 개발하는데 있어서 원시코드 편집기에서 코드를 직접 입력하는 텍스트 지향 프로그래밍 방식뿐만 아니라 폼 디자이너, 오브젝트 인스펙터, 프로젝트 관리자 등과 같은 기능 컴포넌트들을 이용하는 컴포넌트 지향 프로그래밍 방식을 함께 지원함으로써 최근의 RAD 도구들의 장점을 구현하는 것이 가능해질 것이다.

향후 연구로는 주요 목표인 인터넷 환경에서 모빌 코드 기반의 클라이언트-서버 프로그램을 자동 생성해 주는 도구를 개발하기 위하여 현재 진행된 연구결과인 기능 컴포넌트들의 세부적 기능들을 보완 및 추가하고 이를 위한 JCode를 더욱 정형화시켜 IDE의 상부구조를 완성하는 것이다. 그리고 인터넷 및 인트라넷 환경에서 본 도구를 이용한 사용자 클라이언트-서버 응용 프로그램의 속도향상을 위하여 CORBA의 미들웨어와 연동되는 분산 객체 기술의 이용과 클라이언트-서버의 인터페이스 자동 분할 및 생성 기법을 개발하여 지원하는 것이 필요하다.

참 고 문 헌

[1] Ken Arnold and James Gosling, *The Java™ Programming Language*, Addison Wesley, 1996.

[2] James Gosling et al., *The Java™ Application Programming Interface*, Vol.1 & 2, Addison Wesley, 1996.

[3] Robert Orfali, Dan Harkey, and Jeri Edwards, *The Essential Distributed Objects-Survival Guide*, John Wiley & Sons, Inc., 1996.

[4] Robert Orfali and Dan Harkey, *Client/Server Programming with JAVA™ and CORBA*, John Wiley & Sons, Inc., 1997.

[5] Java™ Beans Specification(http://java.sun.com/beans).

[6] *The Java™ Language Environment*, Sun Microsystems, Inc., Oct. 1995.

[7] O'Neal and Miles, *AWT Programming for Java*, Subsidiaries of Henry Holt and Company, Inc., 1996.

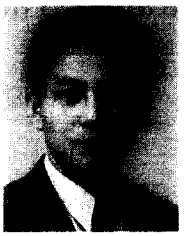
[8] Geary and McClellan, *graphic Java™*, Sun Microsystems, Inc., 1997.

[9] Chris Laffra, *Advanced Java™*, Simon & Schuster Company, 1997.

[10] *Java™ Object Serialization Specification*, Sun Microsystems, Inc., 1997.

[11] *Java™ Core Reflection API and Specification*, JavaSoft, 1997.

[12] Cobb, R. H., "In Praise of 4GLs," *Datamation*, pp.13-18, Jan. 1990.



유 철 중

cjyoo@moak.chonbuk.ac.kr

1982년 2월 전북대학교 전산통계학과 졸업(이학사)

1985년 8월 전남대학교 대학원 계산통계학과 졸업(이학석사)

1994년 8월 전북대학교 대학원 전자계산학과 졸업(이학박사)

1982년 9월~1985년 3월 전북대학교 전자계산소 조교

1985년 4월~1996년 12월 전주기전여자대학 전자계산과 부교수

1997년 1월~현재 전북대학교 자연과학대학 컴퓨터과학과 전임강사

관심분야 : OOSE, HCI, Distributed Objects Computing, GIS, Multimedia, Cognitive Science 등임.