제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

비프로그램 기반 응용 프로그램 시스템 설계

오은진*, 김치수*
*공주대학교 컴퓨터공학과

e-mail : oej73@kongju.ac.kr

Nonprogramming-based Application Program System Design

Eun-Jin Oh*, Chi-Su Kim*
*Dept. of Computer Engineering, Kong-Ju National University

요 약

소프트웨어 개발에 있어 프로그래밍의 어려움과 다양한 사용자 요구 사항의 빈번한 변화로 인한 시스템 설계와 구현사이의 문제는 항상 있어 왔다. 이러한 문제를 해결하기 위해 비프로그램 기반 응용 프로그램 시스템 설계에서는 개발자만이 시스템 소프트웨어를 수정하거나 구현하던 것을 최종 사용자에게 유연성을 증가시켜 시스템 설계와 구현사이의 문제점을 실행 시간(run-time)에 해결하기 위한 것이다.

1. 서론

정적, 개발자 위주의 소프트웨어 개발의 프로그래밍기반 메커니즘(programming-based mechanism)인 기존 방법과 대조적으로, 비프로그램 기반 응용 프로그램 시스템은 상대적, 동적, 사용자 위주의, 비프로그래밍 기반 소프트웨어 개발 메커니즘(non programming-based mechanism)이다. 다시 말해서 소스코드를 직접 코딩한다기보다는 동적 메소드를 통하여 실행 시간에 시스템 설계의 메타데이터를 편집해서 실제로 응용 프로그램을 구축하거나 수정하는 것이다.

따라서 비프로그램 기반 응용 프로그램 시스템은 개인적 사용자에게 동적으로 프로그래밍의 필요 없이 응용 프로그램을 커스터마이징하는 방법으로 소프트웨어 개발의 유연성을 증가시킨다.

본 논문에서 비프로그램 기반 응용 프로그램 시스템의 주요한 아이디어는 다음과 같다.

① 같은 영역에 있는 비즈니스 응용 프로그램을 공통의 지속적인 핵심 비즈니스 로직과 화면표시 로직으로 나누어 인식한다.

② 비즈니스 응용 프로그램의 시스템 설계를 메타모델로 추출해서 비즈니스 로직은 클래스 메타모델로, 화면표시 로직은 GUI 메타모델로 만들어 지속적인 메타데이터로 메타데이터베이스에 저장한다.

③ 시스템 설계의 메타데이터를 최종 사용자가 편집할 수 있게 함으로써 다른 특별한 응용 프로그램 없이 커스터마이징한다.

④ 시스템은 메타데이터를 해석해서 실행 시간에 동적으로 코드를 변환하고, 곧바로 결과를 사용자에게 보여준다.

2. 관련연구

2.1 메타모델링

객체 모델링 기술을 표현하는 메타모델링의 의미는 모델의 모델을 만들기 위한 것이다. 모델을 만드는 항목은 상위 단계 모델의 항목으로 나타나는 인스턴스뿐만 아니라 템플릿의 특징을 가지고 있다.

본 논문에서 메타모델은 주로 meta-class, meta-attribute, meta-method, meta-relationship으

로 이루어지며 다양한 시스템 설계 모델을 정의하고 저장하는 전반적인 개념으로 사용한다.

2.2 메타데이터

메타데이터에 대한 가장 단순하면서도 유용한 정의는 "데이터에 대한 구조화된 데이터"이다. 메타데이터란 어떠한 객체나 자원(물리적인 것이든 전자적인 것이든 상관없음)에 대한 서술 정보로 메타데이터라는 말 자체는 상대적으로 새로운 것이지만 그 바탕이 되는 개념은 수집된 정보들을 조직화하기 시작했던 것만큼 오래 되었다고 할 수 있다[1, 2, 3].

DESIRE project[4]에서는 메타데이터를 "객체의 존재와 특성을 보다 더 확실히 알고자 하는 잠재적인 사용자들로부터 객체를 해방시키는 것과 관련된 데이터"라고 기술하고 있다.

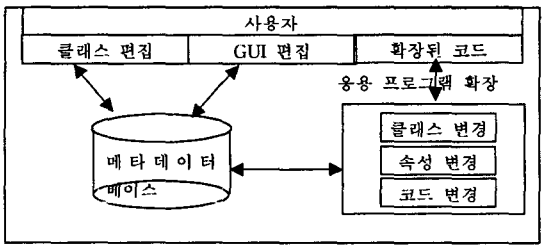
메타데이터는 실제 데이터를 표현하기에 정확해야 하고, 개발자들이나 시스템에 의해 사용되기 쉬우며 융통성이 있어야한다

본 논문에서는 시스템 설계 정보를 영구적인 데이터로 저장하기 위해서 메타데이터를 사용한다.

3. 시스템 설계

본 논문의 메타데이터 베이스는 시스템 설계에 대한 정보를 저장하는 데이터베이스로서 시스템 객체 모델과 시스템 GUI 설계의 메타데이터를 저장하고 관리하는데 사용된다.

데이터베이스에 저장된 시스템 설계의 메타데이터는 데이터베이스 테이블 생성을 지원하는데 사용된다. 사용자는 비즈니스 로직을 다루는 클래스 편집, 화면 설계를 다루는 GUI 편집을 이용해 새로운 클래스를 생성함으로써 메타데이터 베이스 테이블에 추가나 변경을 할 수 있다.

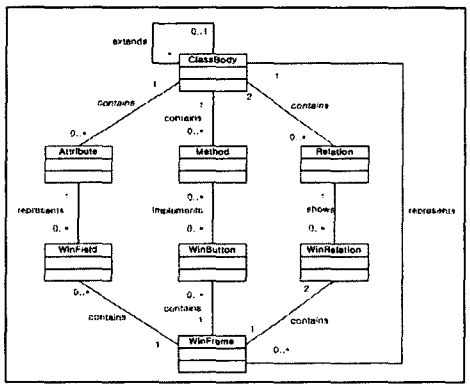


(그림 1) 전체 시스템 구성도

3.1 메타모델 설계

본 논문의 시스템을 위해 시스템의 비즈니스 로직을 담당하는 객체 모델을 기술하는데 사용되는 부분과 이 객체 모델에 기초한 시스템의 화면표시로직인 GUI 모델을 기술하는데 사용되는 두 부분으로 설계하되 독립적이지 않고 통합된 형태가 되도록 설계하여 시스템의 객체 모델과 GUI 모델이 상호 연동되어 동작하도록 함으로써 개별적으로 독립된 작업을 수행하면서 보다 많은 융통성을 부여하도록 한다.

다음 (그림 2)는 본 논문에서 제시하는 메타모델로 시스템 객체 모델을 기술하는데 사용되는 4개의 클래스(ClassBody, Attribute, Method, Relation)와 GUI 설계를 위해 사용되는 4개의 클래스 (WinFrame, WinField, WinButton, WinRelation)로 구성되어 있다.



(그림 2) 시스템의 메타모델

3.2 메타데이터 베이스 설계

데이터베이스 테이블에서 각각의 항목은 대응되는 메타 모델의 클래스의 속성과 거의 똑 같다. 클래스의 속성과 테이블 항목의 차이점은 이름에서 다르다. 본 논문에서 각각의 테이블의 기본 키와 외래키의 정의는<표 1>에서 보여주고 있다.

4. 응용 프로그램 확장 기법

4.1 속성 변경

비프로그램 기반 응용 프로그램 시스템의 설계에서 실행 시간에 객체에 새로운 속성을 추가하거나 속성의 내용을 변경하기 위해 모든 속성은 해쉬 데이터 구조(hash data structure)를 사용한다. (그림 3)은 자바의 해쉬테이블에 클래스의 속성을 어떻게 저장하는가를 보여준다.

이 코드에서 해쉬테이블은 메타데이터의 속성과 동등한 키(Key)를 사용해서 속성을 인덱스한다. 해쉬테이블에 저장된 속성의 값은 총체적 데이터 타입(Object)으로 정의된다. 이것은 다른 타입(String or

<표 1> 메타데이터 테이블의 기본 키와 외래 키

테이블 명	기본 키(Primary Key)	외래 키(Foreign Key)
ClassBody	className	
Attribute	className, attributeName	ClassBody table과 관련된 className
Method	className, methodName	ClassBody table과 관련된 className
Relation	className, relationName	ClassBody table과 관련된 className
WinFrame	className, winName	ClassBody table과 관련된 className
WinField	className, winName, attributeName	WinFrame table과 관련된 assName과 winName Attribute table과 관련된 className과 attributeName
WinButton	className, winName, buttonName	WinFrame table과 관련된className과 winName
WinRelation	className, winName, relationName	WinFrame table과 관련된className과 winName Realtion table과 관련된 className과 relationName

Integer)으로 타입캐스트(typecast)될 수 있다.

```
class AClass{
        private HashTable Properties=new HashTable();

        public void setProperty(String key, Objectvalue){
                properties.put(key, value);
        }
        public void getProperty(String key, Objectvalue){
                properties.put(key, value);
        }
```

(그림 3) 해쉬테이블에 클래스 속성 저장하기

setProperty 메소드는 클래스의 기존 속성 을 갱신하거나 새로운 속성을 추가하는데 사용하고 getProperty 메소드는 속성 값을 검색하는데 사용한다.

속성의 제거나 모든 속성(property key)을 리턴하는 메소드는 클래스의 속성을 다루는 보조 클래스에 추가할 수 있다.

본 논문에서 클래스의 속성을 저장하는 데 해쉬테이블을 사용하는 장점은 사용자가 메타 데이터의 속성을 어떻게 나타내고 액세스하는 지에 대한 선택사항을 제공해서 코드의 복잡성을 데이터로 넘긴다.

또한 사용자에게 실행 시간에 동적으로 클래스의 속성을 변경할 수 있도록 허락함으로써 소프트웨어의 유연성을 상당히 증가시킨다.

4.2 클래스 변경

클래스 변경의 요구시 (그림 4)에서 보여주는 바와 같이 기존방법인 우선 바인딩(early binding) 방법은 적재된 클래스를 실행 시간 까지 알 수 없기 때문에 사용할 수 없다. 이 문제를 해결하기 위해 본 논문에서는 클래스의 지연바인딩(late binding)을 사용한다.

```
import AKnownClass;
.....
AKnownClass aClass= new AKnownClass();
System.out.println(aClass.toString());
```

(그림 4) 자바 클래스의 우선바이딩

지연 바인딩은 알려지지 않은 클래스가 있을 때 시스템이 초기에 컴파일하고 적재한다. 또한 지연 바인딩은 Class type object의 선언을 필요로 하는데 이것은 알려지지 않은 클래스를 저장하기 위해 사용된다.

Class type object는 파일 시스템에서 지연 바운드된 클래스를 위치시키기 위해 forName (classname) 메소드를 호출해서 메모리에 적재시키고 실행 시간에 프로그램 네임스페이스로 바인드한다.

또 지연 바인딩은 Object type object를 선언할 필요가 있는데 지연바운드 된 클래스의 인스턴스를 저장하기 위한 것이다.

이 인스턴스는 지연 바운드된 클래스의 Class type object에서 newInstance()의 메소드의 호출을 통하여 얻어진다. 그런 다음 Object type object는 기본 타입의 하나로 타입 캐스트 될 수 있어서 다른 목적으로 사용할 수 있다.

```
.....
String ClassName="UnKnown";
Class aClass= ClassforName(className);
Object laterBinding=aClass.newInstance();

System.out.println(laterBinding.toString());
.....
```

(그림 5) 자바 클래스의 지연 바이딩

우선 바인딩과 비교하여 지연 바인딩은 실행 시간에 클래스가 동적으로 변경되는 것을 허락하기 때문에 소프트웨어 개발에 있어 상당한 유연성을 가져온다.

또한 우선 바인딩은 참조(reference)로만 사용되기 때문에 시스템이 더 많은 참조를 포함할 때, 더 큰 파일 사이즈와 더 긴 수행시간이 걸리는데 반해 지연바인딩으로 구성한 시스템은 추상적이고 간결하며 시스템에서의 변경은 사용자에게 투명성을 제공한다.

지연바인딩의 단점으로는 추가적 오버로드가 실행시간에 발생할 때 시스템이 느려지는 점과 시스템의 디버깅이 더 어렵다는 문제점을 갖고 있다.

4.3 코드 변경

비프로그램 기반 응용 프로그램 시스템에서 또 다른 중요한 작업은 실행 시간에 메타데이터를 기반으로 생성되거나 변경되는 코드번역이다.

일반적 코드 생성은 소스코드를 쓰거나 고치고, 컴파일하고, 배치하고, 실행하는 단계를 통하여 변경되는 데 비프로그램 기반 응용 프로그램 시스템에서는 이러한 단계가 상당히 부적절하다.

이 문제를 해결하기 위해 본 논문에서는 실행 시간에 코드 변경을 할 수 있는 실행 가능한 인터페이스를 구현하는 클래스를 만드는 것이다.

```
.....
PrintWriter oStream;
.....
try{
        oStream=new
printWriter(newFileOutStream(fileName));
}
catch(IOException){}
.....
//=========== 코드 생성 ============
oStream.println("JFrame aFrame = new JFrame(\" "+title+"\");");
oStream.println("JLabel aLabel = new JLabel(\"
"+labelName+"\");");
.....
```

(그림 6) 일반적인 코드 생성

실행 가능한 인터페이스는 활성된 동안 실행되는 코드 객체에 대한 공통 프로토콜을 제공하기 위해 설계된다.

이 실행 가능한 인터페이스는 run() 메소드로만 선언하는데 쓰레드 클래스를 서브클래스로 하고 run() 메소드를 오버라이드하는 방법과 실행 가능한 인터페이스를 구현하는 클래스를 제공해서 run() 메소드를 구현하는 방법이 있다. (그림 6)은 문자(letter technique)를 동적 코드로 어떻게 번역하는가를 보여준다.

실행 시간에 동적 코드 해석을 얻기 위해 첫 번째로, 실행 가능한 인터페이스를 구현하는 클래스를 선언한다. 두 번째로, run() 메소드를 실행하는 줄을 추가한다. main() 메소드에서 run() 메소드를 실행하는 줄은 프로그램 실행을 시작하는데 사용된다. 세 번째로, 클래스의 지연 바인딩 프로그램의 동적인 것을 실행하는 run() 메소드를 써서 실행 시간에 코드를 수정하는 방식으로 해결할 수 있다. (그림 7)에서 보여주는 것은 실제 구현한 것이 아니라 단지 기술을 보여주기 위한 것이다. 실제 구현되는 프로그램은 좀더 복잡할 것이다.

```
public class GenericApplication implements Runable{
.....
 public static void main(String args[]){
 .....
 run();
}
.....
public void run(){
 .....
 //=========== 지연 바인딩 =======
 .....
 //=========== 코드번역 =========
 .....
 JFrame aFrame = new JFrame(aObject.getProperty("title"));
 JLabel aLabel = new JLabel(aObject.getProperty("label"));
 .....
 }
}
```

(그림 7) 실행 시간을 이용한 코드번역

5. 결론

비프로그램 기반 응용 프로그램 시스템은 시스템의 설계와 구현사이의 문제점을 해결하기 위해 소스코드 필요 없이 다른 프로그래머와 사용자가 시스템을 생성 할수 있도록 하는 방법으로 소프트웨어 개발의 유연성을 증가시킬 것으로 기대된다.

참고 문헌

[1] 한국 더블린 코어 메타데이터, "메타데이터(matadata)란 무엇인가?", at URL: http://www.dublincore.or.kr/faq.htm, 2001.

[2] e-Government Interoperability Framework, http://www.govtalk.gov.uk

[3] Resource Description Framework, http://www.w3.org/RDF

[4] UKOLN Metadata Group. A Review of Metadata: A Survey of Current Resource Description Formats, 1998.