PC 기반 JAVA 프로그램에서 WIPI 프로그램으로의 리엔지니어링[1)]

박성환, 박원주, 박상원
한국외국어대학교 컴퓨터및정보통신공학과
{shpark, wjpark, swpark}@islab.hufs.ac.kr

Reengineering PC-based Java Program to WIPI Program - An Experience Report

Sung-Hwan Park, Won-Joo Park, Sangwon Park
Computer Science & Information Communication Engineering, Hankuk University of Foreign Studies

요 약

최근 모바일 콘텐츠 산업의 급속한 성장으로 모바일 콘텐츠 작성은 새로운 콘텐츠를 작성하기 보다는 기존 PC기반의 프로그램을 모바일 환경에 맞게 변환하는 추세이다. 모바일 환경에 맞게 변환을 할 때 소프트웨어공학 기법을 적용하면 효율적인 리엔지니어링이 가능하다. 우리는 기존 PC기반의 바둑게임 프로그램을 모바일 환경의 프로그램으로 변환하던 중 많은 문제점과 마주치게 되었다. 우리는 문제점을 해결하고, 차후 코드의 재사용성을 고려하여 디자인 패턴을 고려한 리엔지니어링을 했다. 이러한 리엔지니어링을 하면 모바일 콘텐츠 개발자는 차후 PDA와 같은 다른 뷰를 가지는 프로그램으로의 변환도 손쉽게 할 수 있다. 우리는 기존의 패턴 이용하여 코드 재사용성을 높이는 효율적인 리엔지니어링 기법을 제안한다.

1. 서론

최근 핸드폰을 가진 사람을 찾는 것은 어렵지 않다. 핸드폰 산업의 급속한 성장으로 개개인이 대부분 핸드폰을 소유하고 있다. 이에 따라 핸드폰 콘텐츠 산업도 급격히 성장 하고 있다. 어떤 프로그램에 대하여 새로운 콘텐츠를 개발하기도 하지만, 기존의 PC기반 프로그램을 모바일 환경에 맞게 변환하여 제공하는 경우가 많다.

모바일 환경은 PC에 비해 많은 제약사항이 있다. 따라서 기존 PC기반 프로그램은 모바일 환경의 제약사항들을 고려하여 변환이 되어야 한다. 이때, 잘못된 코딩 습관으로 작성된 프로그램은 구조적으로 복잡하며, 가독성이 떨어지고, 재사용 가능한 코드가 적다. 따라서 잘못된 코딩 습관은 프로그램 코드의 재 사용률을 낮추며, 많은 부분의 코드를 수정하거나 새로 작성하도록 만든다.

우리는 기존 PC에서 동작하던 바둑게임 프로그램을 모바일 환경으로 변환을 하면서 많은 문제점과 마주치게 되었다. 바둑게임 프로그램은 매우 잘못된 코딩 습관으로 작성되어 있었다. 우선 표 1에서 보는 바와 같이 전체 클래스의 양이 매우 많았다. 그리고 많은 클래스 중 일부 6~7개의 클래스가 매우 큰 비중을 차지하고 있었으며, 각 클래스마다 불필요한 멤버변수와 멤버함수의 사용량이 많았다. 또한, 주요 기능을 수행하지 않는 클래스와 함수가 역할이 불분명하여 프로그램을 이해하는 것조차 매우 힘이 들었다.

	Total
클래스 수	197 개 (내부 클래스제외)
줄 수	68074 줄

상위 3개 클래스	Client	Lobby	Jungsuk
내부 클래스의 수	11 개	15 개	1 개
줄 수	7143 줄	6215 줄	4281 줄
함수의 수	86 개	88 개	64 개
멤버 변수의 수	162 개	151 개	120 개

(표 1) 기존 PC기반 자바 프로그램 분석

우리는 바둑게임 프로그램을 모바일 환경으로 변환을 하면서 디자인 패턴을 고려한 리엔지니어링(Reengineering)을 하였다. 이를 통해 코드의 양, 클래스의 크기 및 전체적인 프로그램 크기가 작아질 수 있었다. 또한, 도큐먼트-뷰 구조를 적용하여 PDA와 같은 다른 환경으로 이식할 때 코드의 재사용성을 높일 수 있도록 하였다. 그리고 통신 프로토콜을 새로 작성하지 않고 재사용하여 서버프로그램이나 기존 PC프로그

1) 본 논문은 2005년도 한국외국어대학교 학술연구비 지원에 의해서 연구되었음

램의 수정을 가능한 피하였다. 마지막으로 모바일 환경에서 제공하지 않는 라이브러리를 간단하게 작성하여서 라이브러리를 제공하는 다른 환경에서도 코드를 재사용 가능하도록 하였다.

2. 관련연구

WIPI[1]는 Wireless Internet For Interoperability의 약자로 모바일 표준 플랫폼 규격을 말한다. WIPI의 특징으로는 첫째, 단말기의 콘텐츠 개발이 이동통신사에 독립적이며, 둘째, 지금보다 개방적인 표준 플랫폼의 개발이 가능하다는 것이다. 이러한 WIPI는 C언어와 Java언어를 모두 지원하기 때문에 WIPI를 이용하면 콘텐츠 개발이 더욱 용이하다.

디자인 패턴(design pattern)[2]이란 재사용 가능한 객체지향 설계를 만들기 위해 유용한 공통의 설계 구조로부터 중요 요소들을 식별하여 이들에게 적당한 이름을 주고 추상화한 것을 말한다.

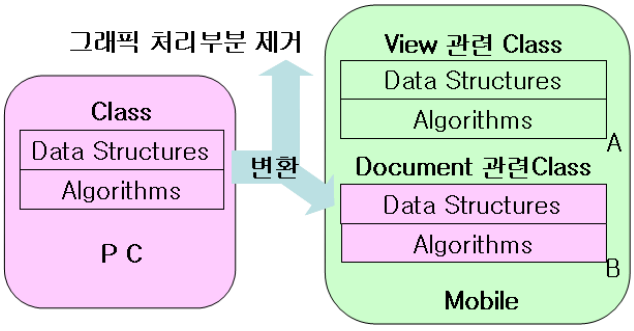
디자인 패턴에는 그 동안 많은 패턴들이 알려져 있지만, 본 논문에서 적용한 중요한 디자인 패턴은 두 가지이다. 첫 번째는 도큐먼트-뷰 구조[3]이다. 도큐먼트와 뷰를 구분지음으로서 알고리즘에 해당하는 도큐먼트위에서 다양한 뷰를 허용할 수 있는 구조이다. 이러한 도큐먼트-뷰 구조는 일반적인 GUI프로그램의 기초적인 구조이다. 이 구조를 통해 전체적인 프로그램의 구조가 간단해지고, 도큐먼트를 재사용함으로써 코드의 재 사용률은 높아진다. 또한 뷰의 재사용 가능한 알고리즘을 다른 뷰에서 이용할 수 있다. 두 번째는 싱글톤(singleton) 패턴으로, 클래스의 인스턴스(instance)는 오직 하나임을 보장하면서 이 인스턴스에 접근 할 수 있는 방법을 제공하는 패턴이다. 싱글톤 패턴은 유일하게 존재하는 인스턴스로의 접근을 통제할 수 있다. 세 번째는 데코레이터(Decorator) 패턴이다. 데코레이터 패턴은 객체에 동적으로 책임을 추가할 수 있게 하며, 기능의 유연한 확장을 위해 상속 대신 사용할 수 있는 방법이다.

3. 코드 재사용성을 높이기 위한 기법

3.1 도큐먼트-뷰 구조

기존 PC에서 동작하던 바둑게임 프로그램은 하나의 클래스에 도큐먼트와 뷰가 섞여 있어서 모바일 환경에 맞게 그대로 변환 하려면 기존의 코드를 재사용하지 못하고 전체 프로그램을 새로 작성 해야만 했다. 그래서 그림 1에서 보는 형태로 기존의 코드의 내용을 도큐먼트와 뷰로 나누는 작업을 하여 새로 작성하는 코드는 기존의 도큐먼트 부분을 약간의 수정을 가해 재사용하고, 뷰는 모바일 환경에 맞도록 새로 작성하는 구조로 바꾸었다.

도큐먼트-뷰 구조를 만들기 위해 기존의 PC 프로



(그림 1) 도큐먼트-뷰 구조로의 변환

그램을 분석해본 결과 크게 6~7개의 클래스에서 중요 기능을 수행하고 있었다. 중요 기능은 크게 4가지로 나누어 볼 수 있었는데, 이 기능을 수행하는 클래스에는 도큐먼트-뷰의 구분이 없이 하나의 클래스에서 모두 처리를 함으로써 굉장히 복잡하고, 작은 버그들이 산재하였다. 또 각 중요 클래스 간에 n:n관계를 가지고 있어서 프로세스의 진행에 일관성이 없었다. 이는 각 클래스를 중복 할당 가능성이 높고, 각 클래스간의 독립성은 보장되기 어렵다.

그래서, 각 클래스마다 도큐먼트와 뷰를 구분하였다. 각 클래스의 코드에 섞여있는 뷰를 제거하고, 도큐먼트를 가려냈다. 이때 뷰에서 필요한 부분을 도큐먼트로 수정하여 삽입하였다. 이렇게 가려낸 도큐먼트는 모바일에 맞게 만들어진 뷰와 통신을 하는데, 서로 통신하는 도큐먼트와 뷰는 서로 연관되도록 하였다.

WIPI 환경에서 뷰는 주로 카드 클래스를 기초로 하여 만들어 진다. 차후 다른 플랫폼으로 바꾸게 되어 뷰가 전환 되어야 할 때, 도큐먼트 부분은 그대로 유지 혹은 약간의 수정만 하여 기존의 코드를 가능한 재사용 할 수 있도록 하면서, 뷰 부분만 바꾸면 된다. 만약 뷰 부분에서도 재사용 가능한 코드가 있다면, 그 코드를 도큐먼트에서 접근할 수 있도록 수정하여 옮기거나, 상속을 통해서 코드를 재사용 할 수 있다.

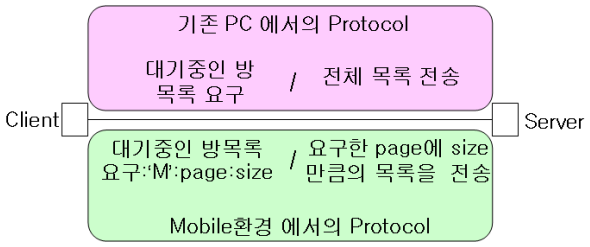
3.2 서버와 클라이언트간의 프로토콜

만약 코드를 재사용하지 않고 프로그램을 새로 작성을 한다면, 기존의 PC프로그램과 통신이 가능한 서버 프로그램과 기존의 프로토콜로 통신은 불가능 하게 된다. 따라서 새로운 프로토콜을 정의하여 사용해야 하므로, 기존에 정상적인 동작을 하는 서버 프로그램을 수정해야 한다. 이러한 서버의 수정을 피하려면 기존의 프로토콜을 사용하면 되는데, 기존의 프로토콜을 사용하는 프로그램을 새로 작성하기 위해서는 복잡하게 코딩 되어있는 PC 클라이언트 프로그램이나 서버 프로그램을 분석하여 각 프로토콜의 동작을 이해하고, 그에 맞는 코드로 작성을 해야 한다.

위의 일들은 매우 비생산적이고 비효율적인 방법이라 할 수 있겠다. 도큐먼트-뷰의 구조로 바꾸면서 기존의 동작하는 코드를 그대로 재사용한다면, 프로토

콜이 그대로 유지되어 매우 효율적으로 변환 할 수 있게 된다. 또, 하나의 서버프로그램으로 다른 플랫폼의 프로그램과 통신이 가능해 진다.

모바일 환경에서는 통신하는 양에 따라 과금하는 제도를 주로 사용하고 있고, 기계자체의 작은 메모리와 낮은 성능을 가지는 문제점이 있다. 따라서 우리는 기존의 프로토콜에 약간의 수정을 가해야 했다. 기존 PC 클라이언트와 서버간의 프로토콜은 PC를 기반으로 하기 때문에 그림 2의 예처럼, 한 번에 많은 양의 정보를 송수신 하고, 수신한 많은 정보를 메모리에 모두 적재하여 사용하는 문제점이 있었다.



(그림 2) 기존 프로토콜의 재사용과 수정

불필요하거나 너무 많은 정보를 송수신함으로써 발생하는 문제를 해결하기 위해 프로토콜의 마지막에 모바일 환경에 필요한 정보를 덧붙이도록 수정을 하였다. 방법은 그림 2의 모바일 환경에서의 프로토콜처럼 기존의 프로토콜에 1바이트 크기의 문자를 추가하여 현재 통신하는 클라이언트가 모바일 클라이언트임을 서버에서 알 수 있도록 하였다. 우리는 'M'을 붙여 모바일 클라이언트임을 표시했다. 그리고 덧붙인 문자 뒤에는 정보의 위치와 수신 가능한 정보의 양 등의 필요한 정보를 추가하였다. 서버에서는 요구한 만큼의 정보를 송신 하도록 수정하였다. 이렇게 수정하는 것은 기존의 프로토콜을 그대로 사용하면서 추가된 정보를 확인하여 서버에서 알맞은 동작을 하게 한다. 또 서버의 수정도 최소로 한다. 그리고 일정량의 정보를 요구가 있는 즉시(On-Demand) 송수신하도록 하고, 송수신한 일정한 양의 정보만을 버퍼에 보관하여 메모리 사용량을 일정하게 유지시켰다.

위와 같은 프로토콜의 재사용과 수정을 통해 다른 플랫폼으로 바뀌더라도 프로토콜에 추가되는 정보만 바꿈으로써 코드와 프로토콜을 재사용 가능하게 한다.

3.3 코드 재사용을 위한 라이브러리 구현

기존의 PC 프로그램은 Java 1.4.02 버전[4]에서 동작하도록 작성되어 있는데, WIPI 1.1 라이브러리[5]에서는 제공하지 않는 라이브러리가 존재하였다. 이로 인해 기존의 코드 중 재사용 불가능한 부분이 존재하였다.

따라서 우리는 모바일 환경에서도 잘 동작할 수 있는 가벼운 라이브러리를 작성하였다. 가벼운 라이브러리를 작성함으로써 모바일 환경에서도 코드의 수정 없이 혹은, 약간의 수정을 통하면 재사용이 가능해진다. 기존의 라이브러리를 사용하던 코드이므로 만약, 다른 플랫폼에서 라이브러리가 제공된다면 크게 수정하지 않고 코드를 재사용할 수 있는 장점을 가진다.

일례로 WIPI에서 제공되지 않는 라이브러리 중 우리는 ObjectInputStream을 다음과 같이 구현 하였다.

```
main function :
void load() {
  Some Object o to implement Serializable;
  FileInputStream    f_in =  new FileInputStream(filename);
  ObjectInputStream o_in = new ObjectInputStream(f_in);
  o = o_in.readObject();
}
```

```
data structure class implement Serializable :
class Data implements Serializable {
  public String _string = "";
  public int     _int     = 0;
  ...

  public void write(ObjectOutputStream out) {
  ...
  }

  public void read(ObjectInputStream in) {
    _string = in.readUTF();
    _int     = in.readInt();
  }
}
```

```
interface :
interface Serializable {
  public void read(ObjectInputStream in);
  public void write(ObjectOutputStream out);
}
```

```
library class :
class ObjectInputStream extends DataInputStream {

  public ObjectInputStream(InputStream in) {
          super(in);
  }

  public Object readObject() {
    Serializable obj;

    // get class name
    String str = super.readUTF();
    if (str.equals("null")) {
        return null;
    }

    Class c = Class.forName(str);
    Object o = c.newInstance(); // make instance
    obj = (Serializable)o;

    // call read function in serializable object
    obj.read(this);

    return o;
  }
}
```

먼저 어떤 Serializable한 오브젝트 o를 ObjectInputStream을 통해서 읽으려고 한다. Serializable 인터페이스는 read와 write 함수를 정의하도록 구현하고, Serializable을 구현하는 데이터 클래스는 인자로 받은 스트림에 자신의 내부 정보를 read, write하도록 구현한다. ObjectInputStream은 저장되어 있는 클래스를 읽어서 Serializable이 구현된 클래스의 인스턴스를 생성한 뒤, 그 클래스의 read함수를 호출하여 내부 정보를 읽어 인스턴스에 저장하게 하고, 그 인스턴스를 반환하는 방식으로 동작한다.

우리는 ObjectInputStream외에도 여러 가지 라이브러리를 만들어서 제공함으로써 코드의 재사용성을 높일 수 있도록 하였다.

3.4 자원 관리자(Resource Manager)의 추가

모바일 클라이언트 프로그램에서 또 하나의 요구사항은 다국어를 지원하는 것이었다. 기존 PC기반 클라이언트 프로그램에는 다국어를 지원하기 위해서 필요한 곳에 하드 코딩(hard coding)이 되어 있었다. 때문에 문장을 수정하고 싶을 때에는 하드 코딩이 된 곳을 찾아 직접 수정을 해주어야 하는 문제점이 있었다. 이를 해결하기 위해 지원하는 언어에 대한 통합적인 관리를 하는 자원 관리자를 구현하였다.

자원 관리자는 싱글톤 패턴을 사용해서 프로그램 내에서 유일함을 보장받는다. 그리고 다른 도큐먼트나 뷰에 대해 완전히 독립적이게 작성하여서, 코드의 재사용이 가능하도록 하였다.

3.5 뷰의 재사용성

도큐먼트-뷰 구조로 변환 하면서 모바일 프로그램의 뷰는 새로 작성해야 했다. WIPI 라이브러리의 뷰는 하나의 카드에 여러 가지 컴포넌트를 그리도록 설계되어있다. 우리는 카드를 데코레이터 패턴으로 동작하도록 사용하였다. 카드를 여러 장 쌓아서 아래쪽 카드에서는 메뉴를, 중간카드에서는 바둑판을, 최상위 카드에서는 바둑돌을 그리도록 하고, 각 카드는 데코레이터 패턴의 동작처럼 자신이 필요한 부분만 그리고 상위의 카드를 그리도록 한다. 때문에 같은 라이브러리를 사용하는 경우 수정이 필요한 카드만 다시 만들면 되므로 뷰의 코드도 재사용 가능하게 되었다.

4. 결과물

	Total
class 수	73 개 (내부 class 없음)
line 수	18009 줄

상위 3 클래스	Baduk	Client	Jungsuk
내부 class	·	·	·
line 수	2710 줄	2614 줄	1873 줄
function 수	58 개	57 개	51 개
Member 변수 의 수	53 개	66 개	73 개

(표 2) 모바일 프로그램 분석

PC 기반 프로그램에서 약 7만 줄이던 코드가 약 2만 줄의 코드로 줄어든 것은 불필요한 함수와 변수를 제거하고, 도큐먼트-뷰 구조로 바꾸면서 꼭 필요한 클래스만 사용하여 클래스의 수를 줄였기 때문이다.

5. 결론

최근 핸드폰의 보급이 급성장함에 따라 모바일 콘텐츠 산업도 같이 급성장하고 있다. 그래서 기존 PC

PC 기반 클라이언트 모바일 클라이언트

(그림 3) PC기반 클라이언트와 모바일 클라이언트 화면

기반의 프로그램들이 모바일 환경으로 많이 변환되어 가는 추세이다.

우리는 기존 PC기반의 바둑게임 프로그램을 모바일 환경에 맞게 변환하는 과정에서 잘못된 코딩 습관에 의해 발생되는 문제를 알게 되었다. 도큐먼트와 뷰가 섞여있는 구조로 구현되어 있던 프로그램을 도큐먼트-뷰 구조로 변환함으로 해서 구조적으로 간단하고, 견고한 프로그램을 작성할 수 있었다. 이는 추후에 PDA와 같은 다른 뷰를 가진 프로그램으로 변환할 때 코드의 재 사용률을 높일 수 있다. 또한 기존의 프로토콜도 그대로 유지되므로 기존에 동작중인 서버의 수정을 최소한으로 한다. 그리고 모바일 환경에서 제공되지 않는 라이브러리를 간단하게 직접 작성함으로써 코드의 재 사용률이 더욱 높아졌다. 또, 자원 관리자를 독립적으로 구현해서 통합적인 자원 관리와 다국어 지원 등에 대해서도 코드의 재사용이 가능해졌고, 마지막으로 뷰를 구현할 때에도 데코레이터 패턴을 사용하여 재사용성을 고려하였다.

이와 같은 리엔지니어링 기법을 통하여 PC기반 프로그램을 모바일 기반 프로그램으로 변환한다면, 모바일 기반 프로그램에서 다른 기반으로 변환할 때 더 효율적이고 재 사용성이 높은 변환이 가능하다.

참고문헌

[1] 박수원, 안은석, 이경철 "위피 모바일 프로그래밍" 한빛미디어, 2003

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides "Design Patterns - Elements of Reusable Object-Oriented Software" Addison Wesley, 2002

[3] Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., & Stal, M. "Pattern-oriented Software Architecture." John Wiley & Sons, 1996

[4] Java J2SE 1.4.2 Document Page, http://java.sun.com/j2se/1.4.2/docs/api/index.html

[5] KTF WIPI 1.1 Document Page, http://hackereyes.hufs.ac.kr/java_doc/ktf/