AOP를 이용한 모바일 소프트웨어의 변환과 적응 방안

이선미○ 서광익 최은만
동국대학교 컴퓨터공학 전공
{lovesm, bradseo, emchoi}@dongguk.edu

Translation and Adaptation Technique for Mobile Software Using AOP

Sun Mi Lee○, Kwang Ik Seo, Eun Man Choi
Dongguk University, Department of Computer Engineering and Science

요 약

소프트웨어가 실행되는 인프라와 물리적인 환경이 바뀜에 따라 변경이 필요한 경우가 갈수록 많아지고 있다. 특히 사용자 인터페이스 방식이나 플랫폼이 바뀌는 모바일 소프트웨어의 경우 설계 과정에 쉽게 전환할 수 있는 융통성을 고려하지 않기 때문에 레거시 프로그램을 새로운 환경에 맞게 변환하거나 동적으로 적응시키기 위한 기술이 필요하다. 본 논문에서는 분리된 관심사를 정의할 수 있는 AOP(Aspect-Oriented Programming) 기술을 이용하여 새로운 사용자 인터페이스 방식이나 새로운 플랫폼에 적응시키는 방안을 제안한다. 이 방법으로 적응이 어려운 레거시 시스템을 동적으로 적응시킬 수도 있으며 재사용할 수도 있다.

1. 서 론

개발된 소프트웨어들이 적응을 전제로 설계된 것이 아니어서 레거시 소프트웨어를 변환하거나 환경에 맞게 적응시키는 작업이 많이 요구되고 있다[1]. 서비스를 업그레이드 하거나 보안을 강화하는 등의 의도로 레거시 시스템을 수정하여야 하는 경우가 많은데 그 때마다 광범위한 레거시 시스템을 이해하고 변경하는 일은 매우 비효율적이다.

예를 들면 스마트폰의 확산과 함께 모바일 소프트웨어의 실행 환경은 빠르게 바뀌고 있다. 스마트 폰이 나오기 이전에도 모바일 소프트웨어는 BREW, GVM, JAVA, WIPI 등 다양한 종류의 모바일 플랫폼을 사용하였기 때문에 모든 플랫폼에 소프트웨어를 제공하기 위하여 한 플랫폼에서 개발된 소프트웨어를 다른 플랫폼으로 실행할 수 있도록 변환하는 작업이 필요하게 되었다.

최근 선풍적인 인기를 끌고 있는 아이폰으로 확산된 터치폰 인터페이스도 레거시 소프트웨어에서 적응해야 하는 환경 중의 하나이다. 키패드의 입력으로만 사용자 인터페이스를 설계한 모바일 소프트웨어는 터치스크린 인터페이스를 고려하지 않았기 때문에 이런 환경에 적응시키기 위한 방안이 필요하다.

환경 변화에 의한 소프트웨어의 변환과 적응은 크게 두 가지의 접근 방법이 연구되었다. 하나는 모델 분석을 기반으로 한 방법이다. 이 방법은 레거시 시스템의 원시코드 모델을 생성하고 적응하려는 목표 시스템의 모델로 변환하는 것이다[2]. 즉 역공학에 의하여 시스템을 이해하고 의도하는 적응을 위한 새로운 모델을 작성하고 이를 기반으로 변환하거나 코딩, 즉 재작업 하는 방법이다. 이 방법은 개발자의 경험에 많이 의존하게 되고 노력이 많이 든다.

두 번째 방법은 정적 분석 또는 동적 분석을 이용하여 변경이 필요한 부분을 찾고 여기에 자동으로 코드를 생성하거나 수동으로 코드를 추가하는 방법이 있다[3]. 사용자 인터페이스나 환경에 영향 받는 부분을 찾기 위하여 동적인 모니터링 코드를 삽입하여 찾고 이 부분을 수정하거나 코드를 추가한다. 이런 방법은 모델을 리엔지니어링하는 수고는 없지만 원시코드를 확보하고 여기에 어떤 형태로든 수정하고 추가하는 정적인 적응 수준에 머무른다. 즉 자동 생성에 의한 동적 적응은 극히 일부로 제한된다.

AOP는 현재 존재하는 프로그램의 유닛, 즉 클래스 범위를 관통하는 관심사를 찾아 이를 별도로 정의한 후 동적으로 엮여지는(weave) 기술이다. 따라서 원래의 시스템, 즉 레거시 시스템과는 별로로 여러 가지 의도를 어드바이스로 정의할 수 있고 이들이 동적으로 결합된다는 매력으로 동적 적응이 가능하다[4].

이 논문에서는 환경 변화에 따른 모바일 소프트웨어의 변환 및 동적 적응 방안을 제안한다. AOP 기술이 원시코드 수준의 자동 변환 방법으로 적합하며 특히 레거시 시스템을 손대지 않고 동적으로 적응시킬 수 있다는 사실을 실험을 통하여 확증하였다. 2장에서는 모바일 소프트웨어의 일반적으로 사용되는 변환 및 재사용 기법을 설명하고 3장에는 이 논문의 핵심인 AOP를 이용하여 레거시 시스템의 수정 지점을 찾고 어드바이스를 정의하여 동적으로 적응시키는 방안을 설명한다. 4장에는 이 방법을 확증하기 위한 실험을 설계하고 5장에서 결과를 기술한다.

2. 관련 연구

AOP를 이용하여 핵심 관심사를 하나의 모듈로 구현하여 레거시 시스템을 건드리지 않고 소프트웨어를 재사용하려면 다음과 같은 주제가 연관되어 연구되어야 한다.

2.1 모바일 소프트웨어의 재사용 기법

모바일 소프트웨어 서비스 산업에는 다양한 플랫폼들이 존재한다. 각각의 플랫폼에 개발하는 것보다 플랫폼 간

의 변환 작업, 즉 동일한 목적의 재사용은 개발자들의 개발 기간과 비용이 감소되는 장점이 있다. 그러나 재사용을 수작업으로 하게 되면 많은 노력과 시간이 소모되는 문제가 발생한다. 이러한 문제를 해결하기 위해 자동 변환 기법을 사용한다.

[7]에 소개된 소스 코드 레벨 변환 방법은 기존의 소프트웨어 코드를 직접 변환하는 것이며, 언어 변환, 커널 변환, API 변환으로 구성된다. 언어 변환은 기존 플랫폼에서 사용하는 언어를 목적 플랫폼에 맞는 언어로 변환하는 것이다. 커널 변환은 기존 플랫폼의 실행 모델을 목적 플랫폼에 맞춰 변경하는 것으로, 이벤트 처리 변경 등을 예로 들 수 있다. API 변환은 기존 플랫폼에서 제공하는 API를 목적 플랫폼 API로 변경하고, 이 API를 이용하여 콘텐츠용 래퍼를 구현한다. 이 세 가지 구성은 플랫폼의 종류에 관계없이 다양하게 적용이 가능하다는 장점이 있지만, 이러한 작업이 복잡하다는 단점이 있다[5-7].

본 논문에서는 [7]의 소스 코드 레벨 변환 방법으로 이용한 spacewar.mc를 참고하였다. 그 중 변환 코드 삽입을 관심사로 제안한다.

2.2 소스 코드 분석 방법

소스 코드 분석 방법으로는 정적 소스 코드 분석과 동적 소스 코드 분석으로 구분된다. 정적 소스 코드 분석은 실행 가능한 프로그램 자체를 분석하는 것이다. 장점으로는 실행 전에 분석이 가능하여 한 번의 분석 결과를 반복적으로 사용 할 수 있다. 동적 소스 코드 분석은 현재 실행되는 프로그램 부분만 분석하여 정확한 결과 값을 획득할 수 있지만, 한 번의 실행만으로는 모든 경우의 결과를 얻을 수 없어 반복적인 실행과 분석을 필요로 한다. 동적 소스코드 분석 방법에는 모니터링 코드삽입 단계가 있다. 모니터링 코드삽입 단계는 필요한 위치인 소스코드 안에 모니터링 코드를 삽입하는 것이다[7-10].

[7]은 추상구문트리(AST)를 이용하여 정적 분석하며 원시 코드에 활성키 모니터링 코드가 삽입된다. 이 경우 원시 코드가 다른 문장과 연관될 수 있어 위험성이 크다. [7]에서는 이러한 문제를 해결하기 위해 문장을 블록화 했다. 블록화를 하게 되면 실행속도가 느리게 되는 단점이 있다.

본 논문에서는 블록화를 하지 않고 레거시 시스템 코드를 보존하며 동적으로 코드 삽입하는 방법을 제안한다.

2.3 Aspect-Oriented Programming

AOP는 여러 관심사를 독립적으로 구현하여 코드의 재사용을 확대한다. 핵심 관심사에 걸쳐진 AOP는 횡단 관심사를 모듈화 단위인 애스팩트로 지원하고 있다. 애스팩트는 횡단 모듈이 포함되어 있고, 이 횡단 모듈이 들어갈 핵심 모듈의 위치를 지정하고 있다. 이러한 횡단 관심사를 제거하고 싶으면 애스팩트를 포함하지 않고 컴파일 하면 된다.

AOP를 사용할 경우 높은 모듈화로 소프트웨어의 복잡성이 감소되고, 각 모듈에 대한 책임이 명확하여 문제 발생 시 추적이 쉽다. 또한 필요할 때 마다 애스팩트를 추가하고 코드 재사용이 확대로 개발 시간이 줄어들어 개발 비용이 감소한다. 따라서 쉽게 시스템을 개선하고 고객이 요구하는 소프트웨어 제작이 가능하다[11].

본 논문에서는 여러 함수에 걸쳐진 관심사를 크로스 컷팅하여 하나의 모듈로 구현한다. 관심사는 [7]의 변환 코드 삽입의 하나인 활성키 모니터링 코드이며, AOP를 이용하여 앞서 소개된 AOP의 장점들을 활용하는 방법을 제안한다.

3. AOP를 이용한 변환과 동적 적응

기존의 모바일 소프트웨어는 키패드를 이용하여 콘텐츠를 작동하는데, 이는 사용자가 키를 누르면 키 이벤트 핸들러가 해당 키를 인식하여 처리하는 방식이다. 그러나 스마트폰과 같은 터치스크린의 경우는 동작하는 방법과 플랫폼이 달라서 키패드를 위해 개발된 응용 프로그램을 그대로 이용할 수가 없다. 따라서 새로운 플랫폼에 맞도록 대폭 수정하거나 호환을 위한 코드를 삽입해야 한다. 하지만 기존 코드에 새로운 코드를 삽입하는 방식은 수정해야할 모듈이 많을 경우 복잡도를 높일 뿐 아니라 소스 코드 내에서 수정해야 할 위치를 찾기도 어려워진다. 이러한 문제를 해결하기 위해 본 논문에서는 AOP를 이용하여 복잡도도 해결하면서 기존의 키패드 기반의 어플리케이션을 수정하지 않고 터치스크린에서 사용할 수 있는 방법을 제안한다.

3.1 활성키 모니터링

활성키 모니터링은 현재 UI 상태에서 키 이벤트 핸들러에 기능이 정의되어 있는 활성키를 관찰하는 것이다. 키패드용 어플리케이션의 호환을 위해 활성키를 이용하는 부분에 애스팩트를 삽입하여 동적으로 터치스크린용 UI 모듈을 호출하게 한다. 그리고 관찰 결과를 바탕으로 터치폰의 화면에 키패드를 자동으로 생성해 준다. 그림 1은 이러한 과정을 표현하고 있다.

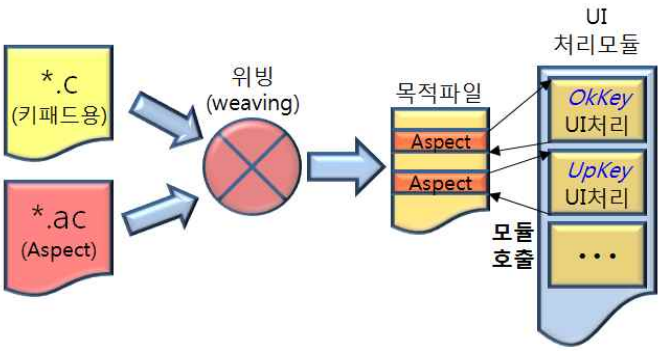


그림 1 활성키 모니터링의 애스팩트 위빙

그림 1은 기존의 키패드용 모듈(*.c)에 동적 활성키 모니터링을 위해 애스팩트(*.ac)를 정의하고, 위빙(weaving) 한 후 목적파일을 생성한다. 애스팩트는 키가 호출되는 시점을 모니터링 하다가 호출되는 시점에 활성키를 처리해주는 터치스크린용 UI를 호출한다. 즉 그림 1의 목적파일의 애스팩트(Aspect)가 동적으로 활성키를 처리하기 위해 필요한 모듈을 호출하게 된다.

3.2 동적 모니터링 적응 애스팩트

키패드용으로 설계된 GVM 플랫폼의 경우 표준 키 매크로 상수는 표1와 같이 정의되어 있다.

키패드용 플랫폼의 콘텐츠는 표 1에 정의된 매크로 상수를 이용하여 키를 인식한다. 따라서 사용된 키를 인식하여 동적으로 이미 정의되어 있는 터치폰용 UI 처리 모듈을 호출하면 키패드용 콘텐츠에서 터치폰용으로 적응이 가능해진다. 애스팩트는 기존의 C 언어로 구현된 원시코드에서 'SWAP_KEY_*'로 정의된 모든 변수의 위치를 파악하고 각 'SWAP_KEY_*'에 해당하는 활성키가 선택될 때마다 이를 인식하고 처리하는 터치폰용 UI 처리 모듈의 호출을 정의함으로 활성키에 대한 동적 적응을 실현한다.

표 1 키패드용 표준 키 매크로 상수

#define	SWAP_KEY_1	0x01
#define	SWAP_KEY_2	0x02
:	:	:
#define	SWAP_KEY_9	0x09
:	:	:
#define	SWAP_KEY_DOWN	0x13
#define	SWAP_KEY_OK	0x14

그림 2는 활성키를 동적으로 모니터링하는 애스팩트를 정의했다. 3번 줄은 애스팩트의 이름을 정의이다. 6번 줄은 표준 키 매크로로 정의된 이름 규칙 중에서 'SWAP_KEY_'로 시작하는 엘리먼트를 접근하는 결합점인 get(SWAP_KEY_*)를 포착하기 위한 교차점을 정의하고 있다. 10번 줄은 monitor(int activeKey) 결합점에서 포착한 매개변수를 이용하여 monitor() 결합점 전에 터치폰 UI 처리를 위한 모듈인 void tuigen_SetActiveKey(int key)를 호출하는 충고를 정의하고 있다. 즉 키패드용 콘텐츠에서 'SWAP_KEY_'로 시작하는 엘리먼트를 접근하기 전에 tuigen_SetActiveKey() 모듈을 호출하라는 애스팩트의 정의이다.

```
1
2  //활성키 모니터링 애스팩트 선언
3  aspect ActiveKeyMonitor
4  {
5    // 표준 키 매크로 포착 포인트컷
6    pointcut monitor(int activeKey) :
7        get(SWAP_KEY_*);
8
9    // 표준 키 매크로 호출 전 충고(advice)
10   before(int activeKey) : monitor(activeKey)
11    {
12      // 활성키 모니터링 모듈 호출
13      tuigen_SetActiveKey(activeKey)
14    }
15 }
```

그림 2 활성키 모니터링 애스팩트

이러한 애스팩트의 정의는 기존의 키패드용 컨텐츠를 수정하지 않으면서도 새로운 터치폰용 플랫폼을 이용할 수 있도록 적응할 수 있는 방법을 제공한다.

4. 실험과 평가

본 실험은 키 이벤트 핸들러 내부에 삽입된 활성키 모니터링 코드라는 관심사를 크로스 컷팅하여 하나의 애스팩트 모듈로 구현하였다.

실험 환경은 표 2와 같다. 우분투 기반에서는 C를 이용한 통합 개발 환경이 거의 없다. 그래서 이클립스를 이용하였고, C언어를 지원하는 애스팩트 패키지인 ACDT 플러그인을 설치하였다. AspectC++은 애스팩트를 위빙할 수 있는 컴파일러이다. 그리고 [7]의 부록 spacewar.mc를 이용하여 간단한 실험을 하였다.

표 2 실험 환경

테스트 환경	테스트 어플리케이션
Ubuntu 9.10	우주 전쟁
eclipse(ACDT플러그인)	소스코드 약 600줄의 테스트용 프로그램
C언어	
AspectC++	

표 3은 우주 전쟁의 3가지 UI상태를 보여주고 있다. 우주전쟁의 디스플레이에 맞춰 실행화면을 갖추었다. 각 실행화면에 대한 레이아웃과 활성키를 나타내고 있다. 터치폰용 플랫폼에서 클릭 이벤트가 일어난다는 전제조건으로 터치 좌표값을 입력하여 실험을 하였다. 각 활성키의 좌표 값은 표3과 같다.

표 3 우주전쟁의 3가지 UI상태

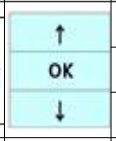
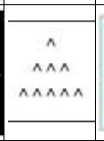
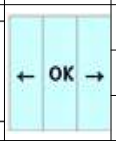
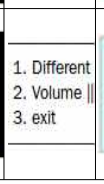
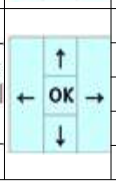
화면	실행 화면	레이아웃	활성키	좌표값 x	좌표값 y
			SWAP_KEY_UP	50	50
			SWAP_KEY_OK	50	150
			SWAP_KEY_DOWN	50	250
			SWAP_KEY_LEFT	50	50
			SWAP_KEY_OK	150	50
			SWAP_KEY_RIGHT	250	50
			SWAP_KEY_UP	150	50
			SWAP_KEY_LEFT	50	50
			SWAP_KEY_OK	150	150
			SWAP_KEY_RIGHT	250	50
			SWAP_KEY_DOWN	150	250

그림 3에서는 활성키를 선언하였다. 그리고 스크린 화면은 높이 300 너비 300으로 정의했다. 제일 중요한 활성키 집합을 구분하기 위해 활성키를 배열로 정의하였다. 이 활성키 집합 배열의 인덱스를 이용하기 위해서 tuigen_base_tkey는 tuigen_tkey_pool에서 정의된 활성키를 찾기 위한 시작 인덱스 배열이다. 그리고 tuigen_size_tkey는 tuigen_tkey_pool에서 정의된 시작

인덱스 위치에서의 활성키 개수 배열이다.

```
//Active Key
#define SWAP_KEY_UP     85
#define SWAP_KEY_DOWN   68
#define SWAP_KEY_LEFT   76
#define SWAP_KEY_PROBE  -2
#define SWAP_KEY_RIGHT  82
#define SWAP_KEY_OK     79
#define SWAP_KEY_CLR    67

//
#define swWidth 300
#define swHeight 300

const int tuigen_tkey_pool[] = {
  SWAP_KEY_OK,
  SWAP_KEY_LEFT, SWAP_KEY_RIGHT,
  SWAP_KEY_UP, SWAP_KEY_DOWN,
  SWAP_KEY_LEFT, SWAP_KEY_OK, SWAP_KEY_RIGHT,
  SWAP_KEY_UP, SWAP_KEY_OK, SWAP_KEY_DOWN,
  SWAP_KEY_LEFT, SWAP_KEY_OK, SWAP_KEY_CLR, SWAP_KEY_RIGHT,
  SWAP_KEY_UP, SWAP_KEY_OK, SWAP_KEY_CLR, SWAP_KEY_DOWN,
  SWAP_KEY_UP, SWAP_KEY_LEFT, SWAP_KEY_OK, SWAP_KEY_RIGHT, SWAP_KEY_DOWN,
  SWAP_KEY_UP, SWAP_KEY_LEFT, SWAP_KEY_OK, SWAP_KEY_CLR, SWAP_KEY_RIGHT,
  SWAP_KEY_DOWN
};

const int tuigen_base_tkey[9] = { 0, 1, 3, 5, 8, 11, 15, 19, 24 };
const int tuigen_size_tkey[9] = { 1, 2, 2, 3, 3, 4, 4, 5, tuigen_MAX_TKEY };

                    :
```

그림 3 헤더 정의

```
aspect Monitor {
  pointcut Preprogress() = "% main(...)";

  advice execution(Preprogress()) : before(){
    ShowLogoUI();
    RunTitle();
    tuigen_InitLayout();
  }

  advice call("% EVENT_GENERAL()")  : before(){

    printf("--> The Layout right now : %d ₩n",tuigen_layout_index);

    int KeyNumber = tuigen_size_tkey[tuigen_layout_index];
    int itemp = tuigen_base_tkey[tuigen_layout_index];

    for(int i = itemp ;i < KeyNumber + itemp; i++)
    {
      tuigen_SetActiveKey(tuigen_tkey_pool[i]);
      printf("tuigen_active_key_set = %c ₩n", tuigen_tkey_pool[i]);
    }
  }

  advice call("% EVENT_GENERAL()")  : after(){
    if(state == ST_TITLE)
      tuigen_layout_index = 4;
    else if(state == ST_RUN)
      tuigen_layout_index = 3;
    else if(state == ST_GAMEOVER)
      tuigen_layout_index = 0;
    else if(state == ST_OPTION)
      tuigen_layout_index = 7;
    //초기화
    tuigen_InitLayout();
  }
};
```

그림 4 활성키 삽입 애스팩트 정의

그림 4는 동적으로 활성키 모니터링 코드를 삽입하기 위한 애스팩트를 정의 한 것이다. 첫 번째는 main함수를 포착하기 위한 교차점을 정의하였다. 두 번째는 main함수가 실행되기 전에 로고와 실행 초기 화면을 보여주고 레이어를 초기화 한다. 세 번째는 터치크스크린 이벤트 핸들러인 EVENT_GENERAL함수이다. 그림 3에서 정의된 tuigen_base_tkey와 tuigen_size_tkey와 tuigen_tkey_pool를 사용한다. 활성키 모니터링 코드를 동적으로 삽입하며 활성키 시작 인덱스 위치부터 활성키의 갯 수를 더한 값을 반복문으로 이용하여 헤더 파일에 정의된 tuigen_tkey_pool과 비교하여 활성키를 출력한다. 그리고 상태가 바뀜에 따라 인덱스를 지정한 후에 레이어에서 터치키 집합을 다시 초기화 해야 한다.

그림 5에서는 먼저 실행시 초기 메뉴 화면을 나타낸다. 실행 초기화면은 애스팩트에서 정의했다. 활성키 집합은 state값에 따라 RunTitleKey함수 실행 전에 {UP, OK, DOWN} 활성키 집합이 삽입되어 출력된다. 그 다음 PLAY를 실행하기 위해 OK좌표값을 입력하였고 바뀐 state값에 따라 {LEFT, OK, RIGHT} 활성키 집합이 삽입된다.

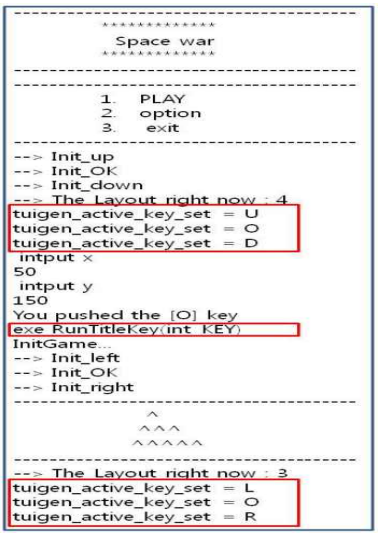


그림 5 실행 초기 화면과 PLAY

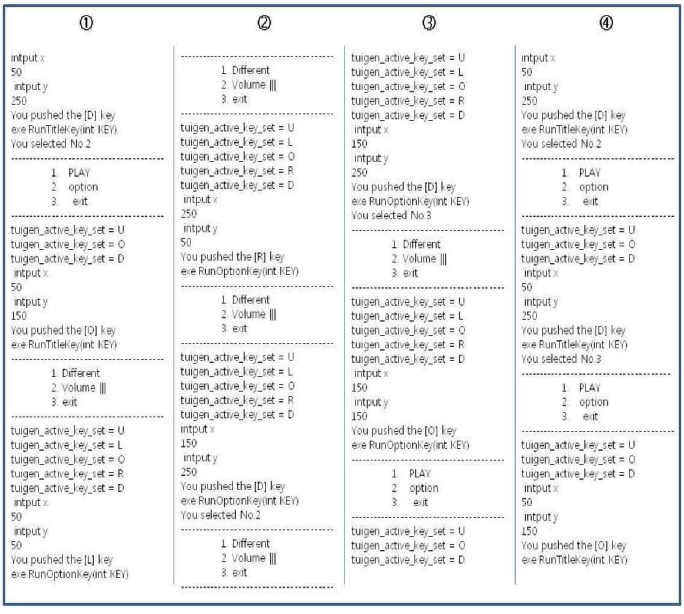


그림 6 option과 exit 실행 화면

그림 6에서 실행 초기 화면은 생략하였다. 그림 6의 실행 순서는 다음과 같다. DOWN을 눌러 타이틀화면의 2

번 옵션으로 가고 OK눌러 2번 옵션이 선택된다. 선택된 옵션이 실행되면서 {UP, LEFT, OK, RIGHT, DOWN}에 대한 활성키들이 삽입되고 출력된다. LEFT와 RIGHT 키 값이 올바르게 실행되는 것을 볼 수 있다. 옵션메뉴에서 두 번의 DOWN을 누르고 OK를 눌러서 exit를 실행하면 다시 처음 타이틀화면으로 빠져나온다. 그 다음 두 번의 DOWN과 OK로 exit가 실행되며 종료된다.

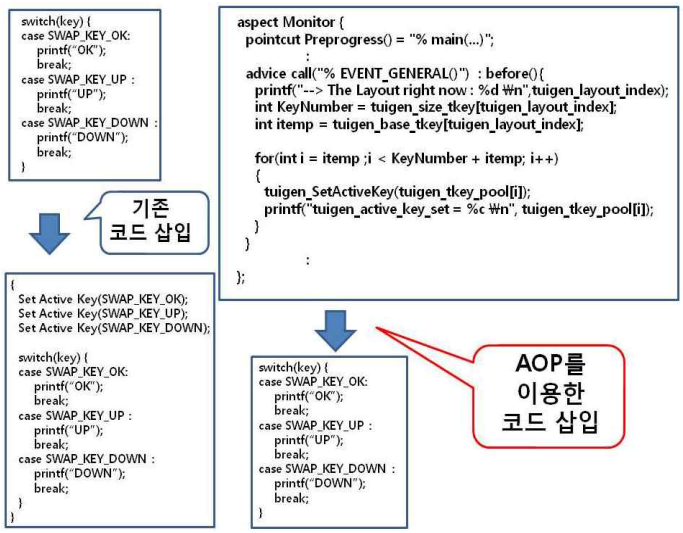


그림 7 기존 코드 삽입과 AOP를 이용한 코드 삽입

[7]은 추상구문트리(AST)로 정적 분석하여 그림 7과 같이 키를 비교하는 if문 또는 switch문을 찾아서 블록화를 하고 활성키 모니터링 코드를 삽입한다. 코드 블록화는 프로그램을 실행할 때 성능이나 효율이 낮아 실행속도가 느리다는 단점이 있다.

따라서 본 실험은 블록화를 하지 않고 원시코드를 보존할 수 있는 애스팩트를 정의하였다. 물론 애스팩트 또한 실행속도가 느려질 수 있지만 모듈화로 인해 관심사에 대한 추적이 용이하여 유지보수가 쉽다는 장점과 관심사 제거시 애스팩트를 제외하고 컴파일하면 된다.

5. 결론

AOP는 현재 구성된 시스템의 구조를 관통하는 관점을 별도의 횡단 관심사로 정의하여 동적으로 결합된다는 면에서 동적 적응에 매력이 있는 기술이다. 따라서 정적인 분석이나 동적으로 레거시 시스템을 분석하기 위한 코드를 잘 모아 캡슐화 할 수 있다. 시스템의 실행 모니터링이나 테스팅, 보안 모니터링 등 여러 가지 응용 문제에 AOP가 적용되어 좋은 결과를 보였다.

이 논문에서는 모바일 시스템을 새로운 환경인 터치 스크린에 적응시키기 위하여 AOP가 효과적인 도구가 될 수 있음을 제안하고 실험을 통하여 보였다. 실험 결과 정적 및 동적인 분석을 위하여 분산되었던 코드를 관점으로 모을 수 있었고 동적으로 웨이빙하여 레거시 시스템의 코드를 잘 보존할 수 있었다. 리엔지니어링을 위하여 활성키를 찾는 문제는 동일한 결과를 내면서 횡단 관심사로 캡슐화 할 수 있고 새로운 환경으로 적응하거나 변환하기 위한 코드도 관심사로 정의 할 수 있다.

참고 문헌

[1] P.K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software", IEEE Computer, vol. 37, no. 7, pp.56-64, 2004.

[2] S. M. Sadjadi and P. K. McKinley,, "Using transparent shaping and web services to support self-management of composite systems", Proc. od the 2nd IEEE International Conference on Automatic Computing, 2005.

[3] 고석훈, 손윤식, 박지우, 오세만, "모바일 게임 콘텐츠의 터치스크린 인터페이스 자동 생성 기법", 정보과학회논문지: 컴퓨팅의 실제 및 레터, 제 15권, 제 1호, pp.866-870, 2009.

[4] J. Zhang and B. Cheng, "Towards Re-engineering Legacy Systems for Assured Dynamic Adaptation", Proc. of International Workshop on Modeling in Software Engineering(MISE07), 2007.

[5] Kyungbo Kang, Donghyun Kang, Changpyo Hong, jongmin Ryu, Joonghoon Lee, Junghan Yoon, and Jeongwoo Jwa, "Development of Conversion Solutions for Interoperability of Applications on Different Mobile Internet Platform," Journal of Korea Contents Society, vol.7, no.4, pp.1-9, 2007. (in Korean)

[6] Sanghun Park, Hyeokju Kwon, Yeonggeun Kim, and Yangseon Lee, "Design and Implementation of GVM-to-MIDP Automatic Translator for Mobile Game Contents," Journal of Game Society, vol.3, no.1,2, pp.5-12, 2006. (in Korean)

[7] 고석훈, 소스 코드 분석을 이용한 터치스크린 UI 자동 생성, 동국대학교 대학원 컴퓨터공학과 박사 학위 논문, 2010

[8] D.D.Cruz, P.Henfiques, and J.S.Pinto, "Code Analysis: Past and Present," Proceedings of the Third International Workshop on Foundations and Techniques for Open Source Softeware Certification(OpenCert 2009), 2009

[9] Sewon Moon and Byeongmo Chang, "A Thread Monitoring System for Multithreaded Java Programs, " ACM SIGPLAN Notices, Vol. 41, no. 5, pp.21-29. 2006.

[10] U. Erlingsson, The Inlined Reference Monitor Approach to Security Policy Enforcement, Ph.D thesis, Cornell University, 2004.

[11] Ramnivas Laddad, AspectJ in Action, pp.28-29, 2003