DOI: 10.3745/KIPSTD.2008.15-D.6.785

Aspect 컴포넌트를 이용한 임베디드 소프트웨어의 모듈 단위 On-The-Fly 테스팅

김 종 필[†] · 홍 장 의[††]

요 약

임베디드 소프트웨어의 테스팅에 대한 다양한 기술 발전에도 불구하고 로봇, 위성 등의 응용 시스템에서는 여전히 빈번한 결함이 발생하고 있다. 이러한 결함의 근본적인 원인은 호스트 상의 테스트 과정에서 발견되지 못한 결함이 타겟 시스템으로 탑재되는 소프트웨어와 함께 내장되기 때문이다. 따라서 이러한 결함으로 인하여 발생할 수 있는 문제를 예방하기 위해서는 타겟 시스템이 동작하는 실 운영 환경에서 소프트웨어의 동작을 테스트하는 접근 방법이 필요하다. 본 연구에서는 Aspect 컴포넌트를 통해 임베디드 소프트웨어의 실행 시점에 기능 및 성능 요소들을 테스트하는 On-The-Fly 테스팅 방법을 제안하였다. 제안한 방법은 호스트상의 테스팅에서 예측하지 못했던 입력 조건에 대한 실제적인 테스트가 가능하며 시스템의 실 운영 과정에서 발생할 수 있는 오동작을 예방하는 기능과 테스트 코드에 대한 높은 재사용성을 제공하는 장점이 있다.

키워드 : Aspect, On-The-Fly 테스팅, 임베디드 소프트웨어

An On-The-Fly Testing Technique of Embedded Software using Aspect Components

Jong-Phil Kim[†] · Jang-Eui Hong[††]

ABSTRACT

In spite of the various techniques on the testing of embedded software, operation failures of embedded systems such as robot or satellite applications, are occurred frequently. The critical reason of these failures is due to the fact that software is embedded into a target system with inherent faults. Therefore, in order to prevent the failure owing to such faults, it needs a technique to test the embedded software which operates in real environment. In this paper, we propose a testing technique, aspect-based On-the-Fly testing that is to test the functionality and performance at real operation time. Our proposed technique gives some benefits of real test of unexpected input conditions, prevention of software malfunction, and reusability of aspect components for the testing.

Keywords : Aspect, On-The-Fly Testing, Embedded Software

1. 서 론

소프트웨어 테스팅은 이미 잘 알려진 바와 같이 완벽하게 수행하는 것은 불가능한 일이며, 잠재된 결함을 얼마나 많이 찾아내느냐가 중요한 관심의 대상이 되어 왔다. 특히 임베디드 소프트웨어에 대한 테스팅은 기존의 일반적인 데스크탑 소프트웨어에 비교하여 테스팅 절차 및 테스트 환경이 정교하고 복잡하다고 할 수 있다[1, 2].

많은 임베디드 소프트웨어, 특히 국방 분야 및 우주 항공 분야의 임베디드 소프트웨어는 단순히 기능 및 성능에 대한 테스팅에 그치는 것이 아니라 실제 환경에서 수행하는 필드 테스트 - 혹은 운영 테스트 - 에도 많은 시간과 비용을 들여 수행하고 있다. 그럼에도 불구하고 최근 F-x 계열의 전투기 추락 사고나 대기권 밖에 버려진 수많은 인공위성을 생각해 볼 때, 아직도 임베디드 소프트웨어 테스팅에 대한 중요성은 강조해도 지나치지 않을 것이다.

본 연구에서는 기존의 호스트 기반 임베디드 소프트웨어 테스팅 방법[3]과 달리 타겟 시스템의 임무 수행 과정에서 자가 테스트를 수행하도록 하는 On-the-Fly 테스팅 기법에 대하여 제안 한다. On-the-Fly 테스트라 함은 타겟 시스템이 실제 환경에서 동작하면서, 자가 진단형 소프트웨어 테스팅을 직접 수행하는 방법을 말하며, 이러한 기법의 제안은

※ 본 논문은 2007학년도 충북대학교 학술연구지원 사업의 연구비지원에 의해 연구되었음
† 준 회 원 : 충북대학교 전자계산학과 박사과정
†† 종신회원 : 충북대학교 전기전자컴퓨터공학부 부교수(교신저자)
논문접수 : 2008년 7월 7일
수 정 일 : 1차 2008년 9월 30일, 2차 2008년 10월 16일
심사완료 : 2008년 10월 16일

기존의 테스팅 과정에서 생성한 테스트 사례가 커버하지 못하는 실제 운영 환경 및 상황에 대해서도 테스트할 수 있도록 하기 위함이다. 제시하고자 하는 테스팅 방법은 Aspect 컴포넌트를 기반으로 하는 테스트 코드를 개발하여 기능 코드와 직조(weaving) 되도록 함으로써, 시스템의 실 운영 과정에서 발생할 수 있는 오동작을 예방함은 물론 소프트웨어 개발과정에서도 기능 코드와 테스트 코드를 분리하여 개발할 수 있는 이점을 제공할 수 있다.

본 논문의 2장에서는 기존의 동적 테스팅 기법에 대한 관련 연구들을 살펴보고, 3장에서는 본 연구를 통해 테스팅하고자 하는 이슈들을 설명하고, 4장에서는 3장에서 설명한 이슈들을 테스팅하기 위한 Aspect 컴포넌트를 주어진 예제에 이용하여 설계한다. 5장에서는 설계된 컴포넌트를 이용한 On-the-Fly 테스팅을 수행하는 절차를 제시하고, 예제 시스템의 테스트 결과를 정리하였다. 6장에서는 결론 및 향후 연구에 대하여 기술하였다.

2. 관련 연구

임베디드 소프트웨어의 테스팅과 관련된 기존의 연구들을 살펴보면 크게 두 가지 측면에서 분석될 수 있다. 첫 번째는 테스트 코드를 기능 코드에 삽입하는 동적 테스팅 기법(BIT, Built-In Test)이며, 두 번째는 Aspect를 이용한 테스팅에 관한 것이다.

2.1 BIT 기반의 테스팅 기법

BIT 기반의 테스팅 기법은 컴포넌트의 내부에 테스트를 위한 기능을 내장하여 컴포넌트의 본래 기능을 제공하는 기능적 인터페이스 이외에 테스트를 위한 인터페이스를 추가로 제공하는 경우이다. 이와 같은 개념을 기반으로 하는 대표적인 연구는 Grob가 연구한 접합점 기반의 BIT 테스팅 기법[4]이다. 이 연구에서는 특정 컴포넌트가 새로운 환경이나 시스템에 설치될 때, 내장된 테스트 기능을 통하여 모듈간 인터페이스의 정합성을 자동으로 테스팅하게 된다. Barbier의 연구[5]도 COTS 컴포넌트의 테스팅에 대한 연구로써, 컴포넌트 기반의 소프트웨어 개발에서 COTS 컴포넌트가 충분히 요구 기능을 충족할 수 있는 가를 Built-in 테스트를 통해 수행한다.

BIT 테스팅 기법에 대한 연구들은 일반적으로 컴포넌트 기반의 소프트웨어 개발에서 많이 이루어지고 있다. 임베디드 소프트웨어에 대한 BIT 테스팅은 호스트 기반의 테스팅에 있어서 테스트 에이전트를 타겟에 탑재하여 수행하는 경우[6]가 대표적이고, On-the-Fly 방법에 의한 테스팅 방법은 거의 존재하지 않으며, 다만 하드웨어 회로의 정상적인 동작을 점검하기 위한 BIT 기반의 Self-Test에 대한 연구들[7]이 임베디드 시스템 영역에서 이루어지고 있다.

2.2 Aspect 기반의 테스팅 기법

기존의 연구에서 Aspect를 이용하는 테스팅은 대체적으로 소프트웨어의 단위시험이나 통합시험 단계에서 수행되는 일반적인 시험 활동의 하나로 간주되어 왔다. 이 과정에서 Aspect은 테스트를 위한 하나의 장비(test harness)로 여겨진다. 이들의 대표적인 연구들은 Bruel [8], Sokenou [9], Lippert [10] 등에 의해 수행되었으며, Bruel은 재사용 컴포넌트가 기능적으로 잘 동작하는 지를 테스트하기 위하여 BIT 기능을 갖는 Aspect를 정의하였으며, Sokenou는 객체지향 단위 모듈에서의 상속성 또는 정보 은익 등의 문제가 올바르게 이루어졌는지를 테스트하기 위한 Aspect를, 그리고 Lippert는 단위 모듈간에 사용되는 문법적인 차이(예를 들면, 센티미터와 피트), 또는 개념상의 차이 등을 검사하기 위하여 테스팅 Aspect를 정의하였다. 이외에도 Filho[11]는 오류 처리를 수행하는 Aspect 클래스를 정의하고, 이를 이용하여 Error Handling Pattern을 정의함으로써, 핵심관심사에 대한 객체지향 모델링을 지원하고 있으며, Pesonen[12]은 테스팅을 수행하는 Aspect를 모바일 운영체제인 심비안에 직조하여 실행시킴으로써, Aspect가 테스트 사례를 받아들여 운영체제의 동작을 테스트할 수 있도록 하였다.

그러나 BIT 기반이나 Aspect 기반의 테스팅 연구들은 전체적으로 실제 운영 과정에서 이루어지는 테스팅이 아닌 호스트 환경에서 이루어지는 테스팅으로 대상으로 하고 있다. 따라서 테스트를 수행하는 Aspect 들은 자체적으로 테스트를 온전하게 수행하기 보다는 테스트 에이전트를 도와주는 역할을 수행하거나, 또는 단순히 테스트 결과를 확정하는 테스트 오라클과 같은 역할을 수행한다.

따라서 본 연구에서는 임베디드 소프트웨어를 대상으로 Aspect 컴포넌트 자체가 독립적으로 테스팅을 수행하고, 또한 테스팅 결과를 바탕으로 소프트웨어의 행위를 제어할 수 있도록 하는 On-the-Fly 방법을 제안하였다.

3. On-the-Fly 테스팅 이슈

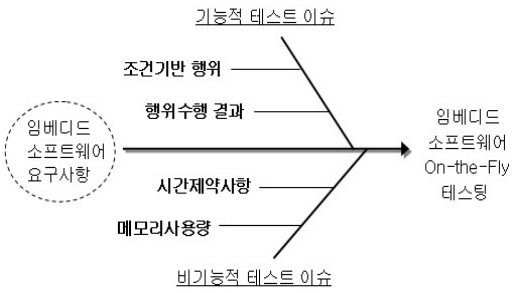
테스트를 수행한다함은 일반적으로 기능적인 요소와 비기능적인 요소의 만족 여부를 검사하는 것으로 이루어진다. 특히 임베디드 소프트웨어의 경우에 있어서는 단순히 기능뿐만 아니라 플랫폼 이식성, 메모리 제약성 등의 다양한 비기능적 이슈들을 테스트하게 된다[13]. 이 중에서도 본 연구에서는 임베디드 소프트웨어의 On-the-Fly 상에서 테스트되어져야 할 이슈들을 (그림 1)과 같이 정리하고, 이들에 대하여 간단히 정의하였다.

3.1 기능적 테스트 이슈

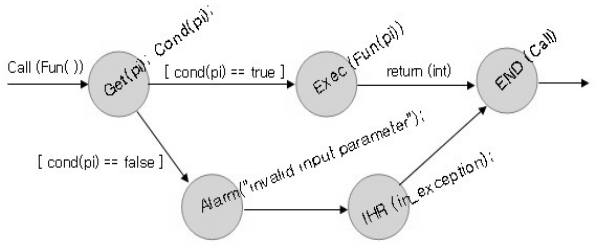
임베디드 소프트웨어의 기능적 테스트 이슈를 어플리케이션의 의존적인 측면에서 유형화 하기는 어려울 것이다. 따라서 본 연구에서는 (그림 1)에서 보는 바와 같이 조건기반 행위 테스트와 행위 수행 영향에 대한 테스트를 On-the-Fly 테스팅 이슈로 구분하였다.

3.1.1 조건 기반 행위 테스트

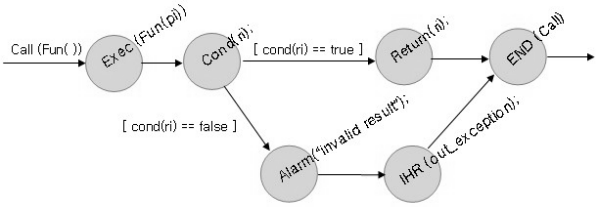
이는 특정 모듈의 실행에 대한 선행조건 또는 후행조건을



(그림 1) 본 연구에서의 임베디드 소프트웨어 테스팅 이슈



(그림 2) 선행 조건 기반의 행위 테스트 명세



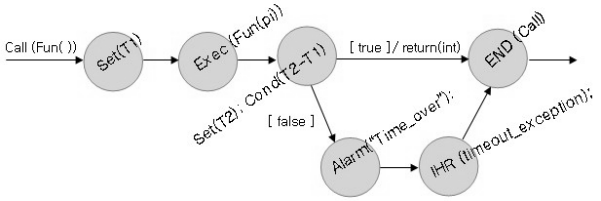
(그림 3) 후행 조건 기반의 행위 테스트 명세

기반으로 테스팅을 수행하는 것이다. 즉 테스트 대상 모듈에서 요구되는 입력이벤트 또는 상태 변수들이 정해진 조건을 만족하는 지 검사하는 것이다. 다음과 같은 기본 함수를 이용하여 조건 기반 행위 테스트의 절차를 (그림 2), (그림 3)과 같이 명세하였다.

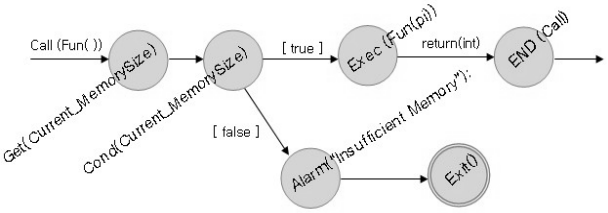
- int Fun(Pi, i=1..n) : n개의 파라미터를 입력받고, 정수형 값을 반환하는 테스트 대상 함수
- Get(x) : 시스템으로부터 입력 변수 또는 상태 변수 x 값을 읽는 함수
- Cond(x) ::= { true | false } : 변수 x의 현재 값이 시스템의 요구사항을 만족하고 있는 지를 평가하는 함수
- Alarm( ) : 특정 문자열을 출력함으로서, 예외 상황을 알리는 함수
- IHR( ) : 예외 상황에 대한 후처리를 수행하기 위한 함수

3.1.2 행위수행 결과 테스트

행위 수행에 대한 결과의 테스트는 먼저 모듈 실행에 대한 상태 변화를 로깅하고, 로그 데이터를 이용한 단위 모듈 실행 결과를 검사하는 것이다. 예를 들면, 임베디드 시스템



(그림 4) 시간제약사항 테스트를 위한 행위 명세



(그림 5) 메모리 테스트를 위한 행위 명세

의 부트 업(boot-up) 모듈이 실행된 후, 정상적인 초기화가 진행되었는지를 확인하기 위해 사용될 수 있다. 이에 대한 테스트 행위 명세는 그림 3과 동일하게 진행된다.

3.2 비기능적 테스트 이슈

3.2.1 시간제약사항 테스트

시간제약사항에 대한 테스트는 스톱워치 형태의 테스트 기능을 제공하는 것으로써, 특정 단위 모듈 혹은 메소드(method)의 실행에 소요된 시간을 측정하여 시간제약사항을 만족하고 있는지를 검사하는 것이다. 시간제약사항을 테스트하기 위한 절차는 다음과 같이 기본 함수를 이용하여 (그림 4)와 같이 명세할 수 있다.

- Set(T) : 현재의 시스템 클럭(time clock)을 저장하기 위한 함수

3.2.2 메모리 사용량 테스트

메모리 사용량에 대한 테스트는 단위 모듈의 실행에서 요구되는 메모리를 충족하는지를 검사하기 위하여 시스템의 가용한 메모리를 점검하기 위한 것으로, 요구 메모리보다 가용 메모리가 많은 경우에 해당 함수를 실행하고, 그렇지 않으면 함수 실행을 종료한다.

위와 같이 제시된 이슈들을 임베디드 시스템의 실제 운영 과정에서 테스트하기 위해서는 적절한 테스트 코드가 개발되어야 한다. 이러한 테스트 코드가 소프트웨어의 기능 코드와 병합(merge)되어 혼재한다면 기능 코드의 개발 및 수정은 물론 재사용도 어렵게 만들뿐만 아니라, 테스트 코드 자체에 대한 확장성도 감소될 것이다. 따라서 Aspect 기반의 테스트 컴포넌트를 별도로 개발해야 하는 필요성이 높아진다.

4. Aspect 컴포넌트의 설계

3장에서 제시된 임베디드 소프트웨어에 대한 테스트 이슈를 On-the-Fly 형태로 지원하고, 또한 기능 코드 및 테스트

코드의 확장성 및 유지보수성을 높이기 위하여 Aspect 개념을 기반으로 하는 테스트 컴포넌트를 설계하였다.

4.1 Aspect 컴포넌트 구성 요소

기존의 AOP 개념으로부터 제시된 Aspect 컴포넌트의 구성요소는 Pointcut, Pointcut designator, 그리고 Advice 등이다[14, 15]. Pointcut은 테스트 기능을 담당할 Aspect와 테스트 대상 모듈이 상호 연결되어지는 접합점을 명시하며, Pointcut designator는 기능 모듈과 테스트 모듈이 접합되는 지점 및 시점을 나타내며, Advice는 테스트 수행 시점을 지정하는 역할을 나타낸다[16, 17]. 본 연구에서도 기존의 Aspect 컴포넌트 템플릿에서 정의하고 있는 기본적인 요소들을 이용하여 컴포넌트를 정의하였다.

참고로 본 논문에서 제시하고자 개발된 Aspect 컴포넌트는 다음과 같은 로봇 응용에서의 요구사항을 갖는 예제 시스템을 대상으로 개발하였다.

> 군사용 정찰 이동 로봇은 설치된 음성 센서들로부터 외부의 소리를 입력받는다. 입력된 소리의 세기를 인식하여 소리의 방향을 식별하고, 해당 방향으로 방향을 전환한 후, 카메라를 이용하여 해당 방향의 물체를 촬영한다. 소리의 인식에서 방향의 전환까지 최대한 빠른 시간 내(최대 1초 이내)에 이루어지도록 하여, 음원의 이동에 대응할 수 있어야 한다.

4.2 기능 행위 테스트 컴포넌트

기능 행위 테스트란 기능 코드가 올바르게 동작할 수 있는 시스템 상태인지, 아니면 기능 코드의 실행 후에 문제없는 시스템 상태를 보장할 수 있는 것인지를 검사하고자 하는 테스트이다. 이를 위해서는 “before” 또는 “after” 타입의 Advice를 사용하여 테스트 대상의 선행조건(pre-condition) 및 후행조건(post-condition)을 검사할 수 있다.

예를 들어 예제 로봇의 음성 센서로 전달되는 입력 데이터가 일정한 값의 범위를 벗어나는 경우 올바르지 않은 시스템 동작을 유발할 수 있기 때문에, 이를 검사하기 위한 테스트용 Aspect를 (그림 6)과 같이 정의하였다. (그림 6)은 AspectJ[18]를 이용하여 개발된 것으로써, 센서 데이터로부터 입력되는 파라미터 decibel의 값이 일정 범위인 80dB와 120dB 범위 내에 존재하는 지를 검사한다. 이는 차량 등의 이동에서 발생하는 음원의 크기에 해당되는 것으로써, 이 범위를 벗어나는 경우는 반응할 필요가 없는 소리이기 때문에 상태 정보를 로깅(기록)하고 해당 모듈이 실행되지 않도록 한다.

```
aspect SensorInputTest {
    pointcut InputRange(int decidel) : Call(void
                Sensor.rvc(int decibel, int hertz);

    before (int decibel) : InputRange (decibel) {
        if ( decibel < 80 ||  decibel > 120 )  {
            Logger.entry ("input range is invalid");
            System.exit(0);
        } else
            Logger.entry ("input range is valid");
    }
}
```

(그림 6) 기능적 요구사항을 테스팅하는 Aspect 컴포넌트

4.3 시간제약 테스트 컴포넌트

임베디드 소프트웨어는 실시간적 특성을 갖는 것이 일반적이다. 따라서 소프트웨어의 시간제약 요구사항을 만족할 수 있을 것 인지를 테스트할 수 있어야 한다. (그림 7)은 센서로 수신된 음성 신호를 인식하여 방향을 결정하고, 이에 따른 구동기의 (방향)전환 명령을 발행(Issue)하는 데 소요되는 시간이 1000ms이내이어야 함을 점검하는 컴포넌트이다. 즉 1000ms 내에 처리하지 못하면 다음 신호의 입력에 의해 전환 명령이 더 이상 유효하지 못하게 될 수 있다. 이 컴포넌트에서에서 “before” advice는 soundDirection() 메소드가 호출되기 전에 타이머를 설정하고 “after” advice는 메소드 실행 후에 시간 초과를 판단하기 위해 사용되었다.

4.4 메모리 제약사항을 테스트하는 Aspect 컴포넌트

한정된 자원을 기반으로 수행되어야 하는 임베디드 소프트웨어는 실행에 필요한 충분한 메모리를 갖지 못하는 경우 오동작의 문제를 유발할 수 있다. 따라서 모듈 실행 전에 얼마의 메모리가 필요할 것인가를 예측하고, 현재의 시스템으로부터 가용한 메모리를 제공받을 수 없다면 실행을 중지하거나 가용 메모리가 확보될 때까지 대기해야 한다. 특정 모듈의 실행에서 요구되는 메모리 사용량은 Eclipse Profiler[19] 등과

```
aspect TimeConstraintTest {
    pointcut timeoutCheck(int decibel) : call soundDirection(int decibel);
    private Timer timer = new Timer();
    long sTime, eTime, tTime;

    before(int decibel) : timeoutCheck(int decibel) {
        sTime = timer.start();
    }

    after(int decibel) : timeoutCheck(int decibel) {
        eTime = timer.stop();
        String str = Integer.toString(eTime - sTime);
        if ( (eTime - sTime) > 1000 ) {
            throw new illegalException("Time is over");
            LCD.drawString("Actual Time:"+str, 0, 0);
            LCD.drawString("Result: invalid", 0, 3);
            System.exit(0);
        } else {
            LCD.drawString("Actual Time:"+str, 0, 0);
            LCD.drawString("Result : valid", 0, 3);
        }
    }
}
```

(그림 7) 시간 제약을 테스팅하는 Aspect 컴포넌트

```
aspect MemoryUsageTest {
    pointcut availmemory() :
        execution(string Controller.*(int decibel, int hertz);

    around( ) : availmemory() {
        Runtime runtime = Runtime.getRuntime();
        String str = Integer.toString(runtime.freeMemory());
        if (runtime.freeMemory() < 19900 ) {
            Logger.entry ("Memory size is Low");
            LCD.drawString("Actual Memory Size:"+str, 0, 0);
            LCD.drawString("Result: invalid", 0, 3);
            System.exit(0);
        }
        else {
            LCD.drawString("Actual Memory Size :"+str, 0, 0);
            LCD.drawString("Reault : valid", 0, 3);
            proceed();
        }
    }
}
```

(그림 8) 메모리 사용량을 테스팅하는 Aspect 컴포넌트

같은 도구를 이용하여 사전에 분석될 수 있으며, 이는 Aspect 테스트 컴포넌트의 Advice에서 검사 조건으로 사용된다. (그림 8)은 메모리 가용량을 테스트하는 Aspect 컴포넌트로써, 시스템의 현재 메모리 사이즈가 19,900 바이트보다 적으면 기능 모듈 시행을 중지시키는 컴포넌트 이다.

5. Aspect 기반의 동적 테스팅

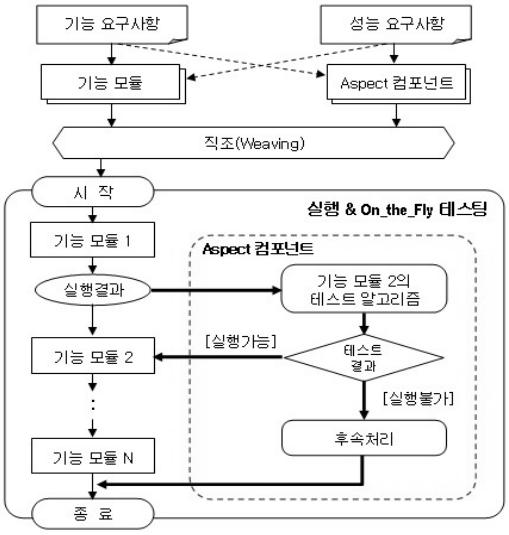
개발된 Aspect 컴포넌트를 이용하여 임베디드 소프트웨어를 테스트하기 위해서는 먼저, 기능 요구사항을 만족하도록 개발된 소스코드와 Aspect가 직조(weaving) 되어 컴파일 되어야 한다. 이렇게 직조되어 생성된 실행 파일은 동작 중에 발생할 수 있는 예외상황을 점검하고 대처할 수 있는 특징을 갖는다.

5.1 On-the-Fly 테스팅 절차

Aspect 컴포넌트는 타겟 환경에서 기능 모듈들이 순차적으로 수행될 때 해당 결합점을 만나면 수행된다. 기능 모듈의 실행을 위해 입력/전달되는 파라미터/이벤트를 통해 해당 모듈이 정확한 동작을 수행할 것인가를 테스트하고, 이 결과를 기반으로 기능 수행 여부를 결정하게 된다. 이렇게 함으로써, 호스트 기반의 테스팅에서 발견하지 못한 예측 불가능한 상황에 대해서 동적으로 대처하게 된다. Aspect가 직조된 임베디드 소프트웨어의 실행 과정은 (그림 9)와 같다.

(그림 9)에서와 같이 기능 모듈과 테스트용 Aspect 컴포넌트는 함께 직조되어 타겟 머신에 탑재된다. 기능 모듈 1이 실행된 후의 실행결과는 기능 모듈 2의 입력으로 사용되게 되는데, 기능 모듈 2의 호출에 결합점이 정의되어 있으면 해당 Aspect 컴포넌트가 기능 모듈 1의 실행 결과를 이용하여 기능 모듈 2에 대한 테스트를 수행한다.

테스트 결과 기능 모듈 2의 실행에 문제가 없다면, 기능 모듈 2가 정상적으로 실행되고, 그렇지 못한 경우에는 후속처리 후 시스템 실행을 종료한다. 이때, 후속처리는 시스템의 요구사항에 의해 결정되는데, 테스트 결과, 결함의 정도가 미비한 경우에는 기능 모듈 1을 재수행할 수 도 있다.



(그림 9) 기능 실행과 On-the-Fly 테스팅 수행 절차

5.2 예제 시스템 적용 결과

앞서 정의한 예제 시스템의 요구사항에 대하여 On-the-Fly 테스트를 수행하기 위하여 4장에서 설계된 3개의 Aspects를 직조하여 실행하였다. 테스트의 편의성 및 단일 요구사항에 대한 정확한 측정을 위하여 각 테스트 이슈의 실험에 대하여 단일의 해당 Aspect만을 직조한 후, 테스트를 수행하였다.

5.2.1 실험 환경 정의

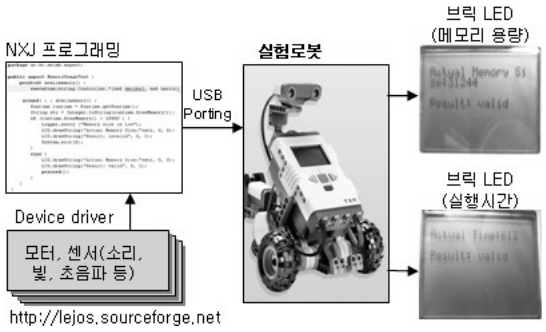
예제 시스템의 적용을 위한 개발 환경은 호스트 환경으로 JDK 1.5.0과 Eclipse 3.1 그리고 AspectJ 1.3.3을 사용하였다. 또한, 메모리 사용량을 추정하기 위한 보조 도구로 Eclipse Profiler 0.5.33[19]을 사용하였다. 타겟 머신은 32비트 ARM7 프로세서와 64K RAM을 탑재한 마인드스톰 레고로봇 NXT #9797을 사용하였다. (그림 10)에서 보는 바와 같이 NXJ 자바 프로그램을 호스트 환경에서 개발하여 USB를 통해 타겟 머신에 포팅하는 방법으로 실험하였다.

4.1절에 정의된 요구사항에 대하여 실험 로봇은 전원이 인가되면, 특정 방향으로 이동을 하다가 소리가 발생하면, 이를 입력받아 이동 방향을 바꾸게 하는 동작으로 실험하였다. 메모리 사용량이나 응답시간에 대한 실험은 잔존 메모리양과 실행시간을 레고 로봇의 브릭 LED에 디스플레이 하도록 Aspect 컴포넌트의 내부에 정의하여 실험 결과를 추출하였다.

5.2.2 기능행위 테스트 컴포넌트의 실행결과

기능 행위에 대한 테스트는 레고 로봇의 음성 센서가 정해진 범위의 소리를 정확히 인식하고 처리하는 지를 테스트하는 것이다. (그림 6)에 지시한 Aspect 컴포넌트를 이용하여 Sensor.rvc(int decibel, int hertz) 메소드의 테스팅 결과는 <표 1>과 같다. 테스팅 결과는 입력 조건에 따른 레고 로봇의 반응 유무로 확인하였다.

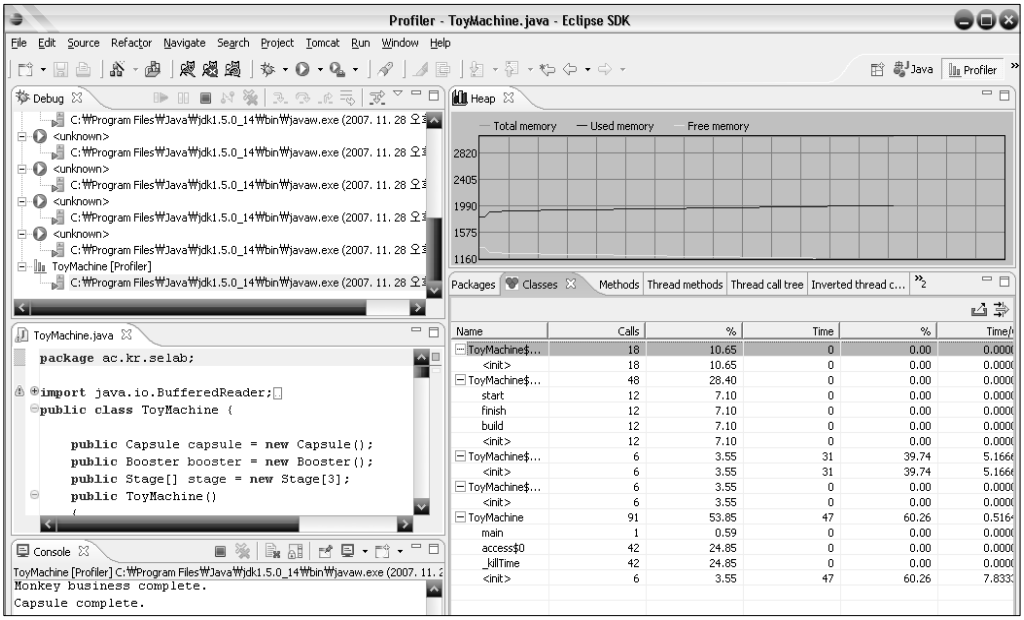
실제 Sensor.rvc(int decibel, int hertz) 메소드에서 입력 파라미터 decibel의 값에 대한 범위 검사를 구현하지 않고, 이를 Aspect 컴포넌트에서 수행하도록 하였음에도 테스트 결과는 <표 1>에서 보는 바와 같이 적합하게 산출되었다. 따라서 특



(그림 10) 예제 시스템 실험 환경

〈표 1〉 기능 테스트 실행 결과

Test Module	Test cases	Input conditions		Expected results	Actual results
		decibel	hertz		
Sensor.rvc()	t1	98	1200	valid - turning	turning
	t2	85	1200	valid - turning	turning
	t3	67	1200	invalid - no change	no change
	t4	104	1200	valid - turning	turning
	t5	71	1200	invalid - no change	no change



(그림 11) Eclipse Profiler에 의한 메모리 사용량 예측

〈표 2〉 메모리 사용량 테스트 실행 결과

Test Module	Test cases	Input conditions		Estimated Memory	Actual Memory	Expected results	Actual results
		decibel	hertz				
Controller.*()	t1	87	1200	19900	32440	valid	valid
	t2	98	1200	19900	31244	valid	valid
	t3	113	1200	19900	28046	valid	valid
	t4	65	1200	19900	14586	valid	invalid
	t5	142	1200	19900	34228	valid	valid

정 기능 모듈에 입력되는 값의 유효성을 점검하고자 할 때, 특히 그러한 유효성이 상황에 따라 달라지는 경우에 Aspect 컴포넌트를 이용하여 기능 모듈 실행의 선행 조건을 테스트 할 수 있다.

5.2.3 메모리 사용량 테스트 컴포넌트 실행 결과

(그림 11)은 Eclipse Profiler[19]를 이용하여 임의의 클래스의 메소드들이 기능 수행 시 필요한 최대 메모리 사용량을 측정하는 모습을 보여준다. (그림 11)의 오른쪽 상단에 특정 모듈의 실행에서 요구되는 메모리 사용량을 그래프로 표현하고 있다.

이렇게 측정된 메모리 사용량은 본 연구에서 제시하고자 하는 Aspect 컴포넌트에서의 테스트 기준으로 사용되게 된다. Eclipse Profiler에 의해 측정된 메모리 사용량을 기준으로 시스템에 잔존하는 메모리의 양이 부족한지 충분한지를 판단하고, 만약 부족하다면 해당 메소드 실행이 불가능하기 때문에 해당 기능 모듈의 실행을 중지시킨다. <표 2>는 Controller 클래스의 메모리 사용량에 대한 테스트 수행 결과이다.

〈표 3〉 시간제약 테스트 실행 결과

Test Module	Test cases	Input condition	Estimated Time(ms)	Actual Time	Expected results	Actual results
		decibel				
SoundDirection()	t1	88	1000	610	valid	valid
	t2	97	1000	600	valid	valid
	t3	115	1000	600	valid	valid
	t4	43	1000	590	valid	valid
	t5	139	1000	610	valid	valid

<표 2>에서 보는 바와 같이 Controller 클래스에 대한 Eclipse profiler의 메모리 사용량은 19,900바이트였으며, 자바 머신의 runtime.freeMemory() 메소드에 의한 가용 메모리 측정 결과는 <표 2>의 Actural Memory 필드에 표시하였다. 테스트의 실행에 있어서 Expected Results는 시스템 메모리 사이즈를 감안할 때, 모두 valid 한 것으로 판단되었으나, Actual results의 경우 테스트 사례 t4의 경우 invalid 한 결과를 얻었다. 그 원인은 특별하지 않으나, t1, t2, 그리고 t3가 정상적으로 실행되어 오는 과정에서 메모리 가비지(garbage)가 발생하였고, t4 실행 과정에서 메모리 부족 현상이 나타난 것으로 여겨진다. 테스트 사례 t4는 Controller 객체의 실행을 종료시키게 되고, 이때 가비지 수집이 발생하여 유효한 메모리 공간이 확보된 것으로 보인다.

참고로 테스트 사례 t4와 t5의 경우는 그림 6에서 제시하고 있는 기능적 요구사항을 만족시키지 못하는 경우이나, 본 논문에서 제시하는 Aspect가 메모리의 충분성만을 점검하기 위한 목적으로 설계/개발 되었으며, 또한 테스트 과정에서 단일 Aspect만을 직조하여 테스트를 수행하였기 때문에 이에 대한 충족 여부만을 고려하여 테스트 결론을 내렸다. 그러나 추후 모든 Aspect들이 처리한 결과를 합하여 필터링하면, 전체 시스템 차원에서의 오류를 확인할 수 있다.

이와 같은 결과로부터, 실제로 valid 하다고 예측되어지는 경우에도 시스템 소프트웨어의 영향에 따라 오류가 발생할 수 있음을 알 수 있었으며, 이에 대한 문제해결을 위하여 Aspect 컴포넌트에 메모리 가비지를 수집하는 메소드를 호출하도록 할 수 있었다.

5.2.4 시간 제약 테스트 컴포넌트 실행 결과

시간제약 사항에 대한 테스트는 실험 로봇이 주행중 소리를 인식하고 바퀴의 방향을 전환하기 시작하는 시점까지의 소요시간으로 산정하는 것이다. 그러나 실제 로봇의 동작 상태에 대한 정확한 시간 측정이 어렵고, 미세한 측정에서의 변화 폭이 커서, 실험 데이터를 테스트 결과로 사용하기에는 부족했다. 따라서 호스트상에서 프로그램 코드에 테스트 실행을 위한 출력문을 추가하여 실행에 필요한 시간을 측정하였으며, 그 결과는 <표 3>과 같다.

<표 3>에서 Estimated Time은 로봇 시스템에 대한 요구사항으로부터 기인하며, Actual Time은 호스트상에서 실행된 시간을 기준으로 표현하였다. 실제 실험 로봇의 운영에 있어서는 다소 차이가 있을 것으로 보이나, 그 결과는 요구사항에서 제시한 1초 보다는 작을 것으로 판단되었다.

5.3 Aspect 기반 테스팅의 효율성

Aspect 기반의 테스팅에서는 테스트 대상 모듈의 수와 무관하게 원하는 테스트 유형 −기능 요구사항 테스트, 또는 비기능 요구사항 테스트−에 대하여 1개의 Aspect 컴포넌트를 개발한다. 단지 Aspect 컴포넌트 내에 테스트 대상 모듈의 접합점(Pointcut)을 선언하는 것으로 테스트 모듈의 개발이 완성된다. 그러나 BIT 기반의 테스팅에 있어서는 테스트 대상 모듈의 수만큼 테스트 모듈이 필요하게 되며 경우에 따라서는 하나의 BIT 모듈에 여러 테스트 유형이 함께 구현될 수 있다. 이를 요약하면 <표 4>와 같이 정의할 수 있다.

본 논문에서 사용한 실험 로봇의 경우, 전체 10개의 모듈 중에서 테스트 대상 모듈이 7개이며, 이에 대하여 3개의 Test Aspects를 개발하였다. 이로부터 우리는 Aspect 기반의 테스팅이 BIT 기반 테스팅보다 최소 4/7의 노력이 감소되었음을 알 수 있다.

6. 결 론

임베디드 소프트웨어의 테스팅에 대한 다양한 기술 발전에도 로봇 등의 응용 시스템이 빈번한 결함 발생 현상을 보이고 있다. 이러한 현상은 개발 소프트웨어 자체가 갖는 결함에 근본적인 원인이 있겠지만, 문제는 호스트상에서의 테스트 과정에 이러한 문제를 발견하지 못하는 것에 있다. 따라서 본 연구에서는 Aspect 컴포넌트를 개발하여 임베디드 소프트웨어의 실행 시점에 기능 및 성능 요소들을 테스트하는 On-the-Fly 테스팅 방법을 제안하였다.

이렇게 함으로써, 호스트상의 테스트에서 간과된 결함을 실행시점에서 찾아낼 수 있으며, 또한 예측하지 못했던 입력 조건에 대하여 실행/운영 도중의 실제 데이터를 활용하

〈표 4〉 테스트 모듈 수의 상대 비교

테스트 대상 모듈 수	테스트 유형 수	BIT 모듈 수	Test Aspect 모듈 수
n개	ℓ개	ℓ*n개, 또는 n개	ℓ개

여 테스트 한다는 장점을 제공한다.

그러나 일반적으로 하드 리얼타임(hard realtime) 시스템의 경우는 Aspect 컴포넌트를 기능 모듈과 직조하여 탑재하는 것이 반응시간을 지연시키는 결과를 초래할 수 있다는 문제를 지적할 수 있다. 그러나 실제 본 논문의 사례에서 개발된 예제에서 보듯이 Aspect 컴포넌트의 크기는 기능 모듈에 비하여 매우 단순하고 짧아서 실행 시간에 크게 영향을 주지 않을 수 있으며, 또한 그 실행시간이 어느 정도 감내할 수 있는 범위 - 대체적으로 1초 이내 - 에 존재하기 때문에 일반적인 임베디드 소프트웨어의 적용에는 문제가 없을 것으로 보인다.

본 연구에 따른 향후 연구로는 Aspect 개념에 대한 확장을 통하여 기능 모듈 자체에 대한 테스트가 아닌 다수의 기능 모듈간에 발생할 수 있는 문제들 - 예를 들면, 데드락, 선점 스케쥴링에서의 데드라인 충족성, 경쟁 상태(race condition) 같은 사항들을 테스트할 수 있는 방법들을 개발하는 것이며, 이들 테스트 컴포넌트를 이용하여 보다 더 복잡한 시스템의 테스팅에 적용하고 그 유용성 및 실용성을 확인하는 것이다.

참 고 문 헌

[1] A. Coulter, "Graybox Software Testing Methodology - Embedded Software Testing Technique," 18th Digital Avionics Systems Conference Proceedings. pp.10-17, 1999.

[2] E. Dustin, 'Effective Software Testing-50 Specific Ways to Improve Your Testing,' Pearson Education, 2003.

[3] M.J. Karlesky, W.I. Bereza and C.B. Erickson, "Effective Test Driven Development for Embedded Software," IEEE EIT'06, East Lansing, pp.382-387, May, 2006.

[4] H.-G. Grob, "Built-In Contract Testing in Component-Based Application Engineering," LOPSTR'02 2002, Spain, pp.87-100, Sept., 2002.

[5] F. Barbier, "COTS Component Testing Through Built-In-Testing," in Testing COTS Components and Systems, edited Sami Beydeda, Springer, pp.55-70, 2005.

[6] I. Pavlova, M. Akerholm, and J. Fredriksson, "Application of Built-In-Testing in Component-based Embedded Systems, ISSTA'06, Portland, pp.51-52, 2006.

[7] T. Sumi and O. Mizuno, "An Effective Testing Method for Hardware Related Fault in Embedded Software," IEICE, Vol.E88-D, pp.1142-1149, 2005.

[8] J. Bruel and J. Araujo, et. al., "Using Aspects to Develop Built-in Tests for Components," UML03, San Francisco, USA, pp.1-8, 2003.

[9] D. Sokenou and M. Vosgen, "FlexTest: An Aspect-Oriented Framework for Unit Testing," QoSA/SOQUA'05, pp.257-270, 2005.

[10] M. Lippert and C. V. Lopes, "A Study on Exception Detection and Handling Using Aspect-Oriented Programming," ICSE 2000, pp.418-427, 2000.

[11] F. Castor Filho and A. Garcia, et. al., "Error Handling as an Aspect," BPAOSD'07, Vancouver, 2007.

[12] J. Pesonen, "Extending Software Integration Testing Using Aspects in SymbianOS," TIAC-PART'06, pp.147-151, 2006.

[13] M. Loghi and T. Margaria, "Dynamic and formal verification of embedded systems," Journal of Parallel Programming, Vol.33, pp.585-611, 2005.

[14] 최재영 외 4인, "관점지향프로그래밍(AOP)의 소개와 응용", 정보과학회지 제 24권, 제12호, pp.21-27, 2006.

[15] R. Laddad, 'AspectJ in Action: Practical Aspect-Oriented Programming,' Manning Publications, 2003.

[16] 김종필, 홍장의, "임베디드 소프트웨어의 동적 테스팅을 위한 Aspect 컴포넌트의 설계", 임베디드공학 추계학술대회, pp.236-239, 2007.

[17] 김태웅, 김태공, "AOSD기반에서 Aspect의 동적 결합을 위한 Connector," 정보처리학회논문지D, 제13-D권 제2호, pp.251-258, 2006.

[18] G. Kiczales, et al., "An Overview of AspectJ," LNCS, Vol. 2072, Springer, pp.327-353, 2001.

[19] http://eclipsecolorer.sourceforge.net/index_profiler.html, 2004.

김 종 필

e-mail : kimjp@selab.chnugbuk.ac.kr
2006년 충북대학교 컴퓨터공학과(학사)
2008년 충북대학교 전자계산학과(공학석사)
2008년~현 재 충북대학교 전자계산학과 박사과정
관심분야 : Aspect 기반 컴퓨팅, 임베디드 소프트웨어, 소프트웨어품질공학

홍 장 의

e-mail : jehong@chungbuk.ac.kr
1988년 충북대학교 전산학과(이학사)
1990년 중앙대학교 컴퓨터공학과(공학석사)
2001년 한국과학기술원 전산학(공학박사)
2002년 국방과학연구소 선임연구원
2002년~2004년 (주)솔루션링크 기술연구소장
2004년~현 재 충북대학교 전기전자컴퓨터공학부 부교수
관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어 품질공학, 소프트웨어 프로세스