제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

임베디드 소프트웨어 테스팅 도구의 설계 및 구현

장선재, 김지영, 손이경, 김행곤
대구가톨릭대학교 컴퓨터정보통신공학부
e-mail : {jsjsid, kimjy, twilly, hangkon}@cu.ac.kr

Design and Implementation of Embedded Software Testing Tool

Seon Jae Jang, Ji-Young Kim, Lee-Kyeong, Son, Haeng Kon Kim
*Dept. of Computer Information & Communication Engineering

요 약

임베디드 시스템의 발달은 다양한 분야로의 보급으로 인해 널리 확장되면서 점차 빠르게 발전되어 왔다. 이러한 빠른 발전은 점차 다양한 기기들의 융합인 컨버전스로 나아가고 있으며, 동일한 분야의 기기에서 시스템은 다르지만 비슷한 성능을 가진 기기들이 등장하고 있다. 이렇게 임베디드 시스템이 다양해지고 비슷한 성능의 제품들이 출시되는 상황에서 임베디드 소프트웨어의 중요성은 더욱 증가할 것이며, 오류를 검사하는 테스팅은 더욱더 중요해 지고 있다.

본 논문은 이기종 환경의 다양한 임베디드 시스템을 지원가능하고, 보다 가볍고 효율적으로 사용가능한 임베디드 소프트웨어 테스팅 도구를 설계하고 구현하고자 한다.

1. 서론

최근 하드웨어의 다양한 발달과 보급은 모든 산업, 생활기기들에 IT기술이 내장되는 "Embedded Everywhere"시대의 도래를 예고하고 있다. 개발되는 기기들은 점차 소형화 되고 있으며, 다양한 기기들이 하나에 합쳐지는 컨버전스가 일어나고 있다. 다양한 플랫폼에서 유사한 성능, 또는 다른 시스템 체계에서 동일한 OS가 사용됨에 따라 기기간의 성능 차는 좁혀지고 있으며, 이에 따라 사용되는 소프트웨어 중에서도 임베디드 소프트웨어의 중요성이 증가되고 있다. 소프트웨어의 중요성 증가는 이를 분석하고 평가하는 테스팅의 중요성 또한 증가되고 있다.

임베디드 기기들은 기존에 사용되는 일반 기기와는 달리 많은 제약이 존재하는데 이는 임베디드 기기가 실시간성, 높은 신뢰성, 상대적으로 적은 메모리와 저전력을 요구하는 특성에 기인한다. 또한 다양한 기기들이 빠르게 시장에 출시되고, 서로 유사한 성능을 가지고 있는 경우가 많기 때문에, 탑재되는 OS와 응용 소프트웨어의 품질과 성능이 더욱 중요시되게 된다.

따라서, 임베디드 소프트웨어에 대한 테스팅은 제한된 임베디드 시스템 상에서 보다 효율적이고 좋은 임베디드 소프트웨어 개발에 기여할 수 있다.

본 논문은 임베디드 소프트웨어를 테스팅 하는 도구를 제안하고자 한다. 2장은 관련연구로서 테스팅에 사용되는 방법들을 소개하고, 3장에서는 임베디도 소프트웨어 테스팅 도구의 설계, 4장에서는 구조와 구현, 마지막으로 5장에서는 본 논문의 결론을 맺는다.

2. 관련연구

2.1 소프트웨어 테스팅 방법

소프트웨어 개발에서 테스팅은 오류를 발견하여 소프트웨어의 품질을 향상시키지만, 간단한 성능을 가진 예전의 임베디드 소프트웨어에 비해 최근의 임베디드 소프트웨어는 크기도 증가하고 더욱 복잡해지면서 모든 경우를 테스트 하는 것은 불가능에 가까워지고 있다.

<표 1> 경로 테스팅

종류	방식
경로 커버리지	모든 경로의 100%실행
문장 커버리지	모든 문장 실행
가지 커버리지	모든 상황에서 모든 가지

소프트웨어의 모든 가능한 경로를 실행하는 것이 불가능해짐에 따라, 테스트의 완전함 정도에 대한 척도를 주기위한 여러 방법들이 개발되었다. 그 방법 중에 관측가능성 방법과 경로 테스팅 방법이 대표적이며, 가장 널리 이용되는 테스팅 방법은 경로 테스팅이다. 경로 테스팅은 크게 경로 커버리지, 문장 커버리지, 가지 커버리지를 예로 들수 있다[1][2].

<표1>은 경로 테스팅의 종류 중 대표적인 것을 나타낸 것이다. 경로 커버리지는 모든 가능한 경로를 100% 실행하는 방법으로 경로 테스팅 방법 중 가장 완벽하지만, 모든 경로를 실행함으로서 많은 수의 경로를 테스트가 필요하며, 경로 중 실행 불가능한 경로가 존재할 수 있다. 문장 커버리지는 프로그램 안의 모든 문장을 실행하는 방법으로, 쉽게 실행될 수 잇는 이점이 있으나 오류가 발생 가능한 상황을 테스트 하지 않고 넘어갈 수가 있다. 가지 커버리지는 각각의 가지에서 발생가능한 모든 가능성을 테스트하는 것으로 문장 커버리지에 비해 약간의 성능 향상이 있을 뿐이다[1].

관측가능성 방법은 출력에 영향을 미치는 부분을 관측하는 것으로, 구문 커버리지와 가지 커버리지를 충족시킬 수 있는 방법이다. 또한, 출력에 영향을 미치지 않는 요소들을 파악하고 테스트 케이스를 추가하여 좀 더 높은 커버리지를 이끌어낼 수 있다.

3. 임베디드 소프트웨어 테스팅 도구 설계

임베디드 소프트웨어는 개발되는 환경과는 다른 플랫폼에서 동작하는 응용 프로그램이다. 따라서 테스팅 도구 또한 지원되는 플랫폼에 맞추어 테스팅할 수 있어야 하며, 보다 제한된 환경인 임베디드 시스템의 사정에 맞는 테스팅 체제가 구축되어야 한다[3].

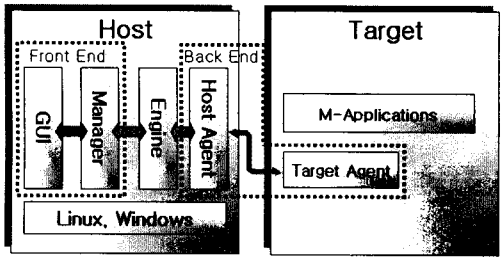
임베디드 소프트웨어 테스팅 환경은 (그림 1)과 같다. 임베디드 소프트웨어를 임베디드 시스템에서 테스팅하는 데는 많은 무리와 제약이 따른다. 일반적으로 이러한 제약을 극복하기 위해서 소스입력, 분석, 테스트케이스 생성 등을 호스트에서 실행하고, 타깃은 테스트케이스를 실행하고 그 결과를 호스트로 보내는 작업을 수행한다. 결과를 받은 호스트는 이를 분석, 정리하여 결과를 표시하게 된다.

임베디드 소프트웨어 테스팅은 우선 테스팅할 소스를 읽어 들이고, 타깃에 맞추어 컴파일 한다. 컴파일된 소스를 분석하여 스크립트를 작성하고 테스트 케이스을 생성한다. 생성된 테스트 케이스를 사용하여 테스트를 수행하고 수행된 결과를 분석하여 보고하는 순서이다.

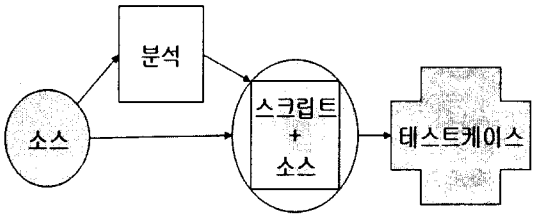
GUI와 Manager은 테스팅 툴에서 중심을 담당하는 부분이라 할 수 있다. 테스팅 전반에 대한 과정을 보여주는데, GUI와 Manager의 Front End는 테스트할 소스를 읽어 들여 다음 단계인 엔진에서 테스트를 수행할 준비 작업을 수행한다. 또 테스팅된 결과를 받아들여 사용자에게 보고할 수 있는 형태로 제공하는 등 다양한 역할을 수행한다.[4]

엔진은 테스팅 툴에서 핵심을 담당하는 부분으로서, (그림 2)와 같이 Front End에서 받아들인 컴파일된 소스를 분석하여 테스트를 위한 스크립트를 생성하고, 이를 기반으로 테스트 케이스를 생성한다. 또한 생성된 테스트 케이스를 사용하여 테스트를 진행하여 여러 가지 평가를 수행하며, 이때 생성된 테스트 데이터들을 Front End로 넘긴다.

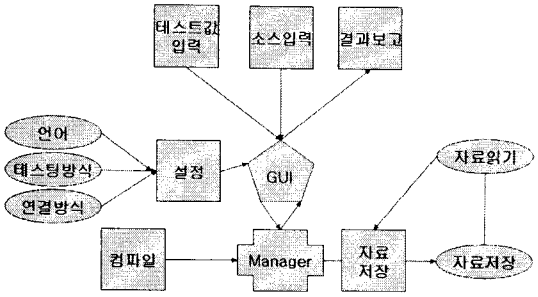
마지막인 Back End는 호스트와 타깃을 연결하는 부분으로, 엔진에서 제작된 테스트 케이스를 타깃에 전송하고 타깃에서 테스팅된 데이터를 호스트로 전송하는 역할을 수행한다. Back End의 경우 다양한 연결방식을 지원할 수 있다.



(그림 1) 임베디드 소프트웨어 테스팅 시스템 구조



(그림 2) 엔진의 구조

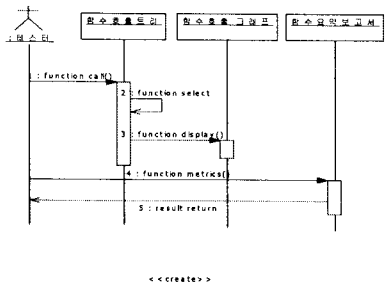


(그림 3) Front End 구성도

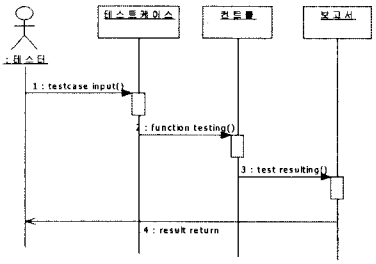
4. 임베디드 소프트웨어 테스팅 도구 구현

(그림 3)은 Front End의 구성과 역할을 나타낸다. Front End에서 주목해야 하는 점은 소스를 읽고 그것을 테스트할 플랫폼에 맞추어 컴파일하는 것이다. 이것은 각각의 플랫폼에 사용가능한 SDK를 이용한다. Front End는 필요한 SDK를 선택하는 기능이 필요하며, 또한 테스트 결과 데이터를 어떠한 방식으로 사용자에게 보고하는가를 고려하여야 한다. 대표적으로 수치화나 가시적 표현 방식이 있다.

엔진의 구현은 크게 2부분으로 나눌 수 있는데, 분석과 연산부분이다. 분석부분은 Front End에서 받은 컴파일된 데이터를 분석하고 스크립트를 생성하는 것이며, 연산부분은 분석부분에서 분석 데이터를 받아 테스트케이스를 생성하고 실행 시키는 부분이다. 분석부분은 크게 명령어를 위주로 하여 하나의 컨트롤 흐름도를 작성한다. 이때 각각의 명령에는 주석형식으로 새로운 코드들을 추가시킨다. 이로서 분석부분에서의 작업은 완료되며, 생성된 분석데이터는 연산부분으로 이동된다. 생성된 분석데이터는 데이터가 호완 가능한 다른 테스팅 툴이나 분석 소프트웨어, 도구 안에서 다양한 테스팅 방법을 사용시에 효과적으로 운영할 수 있게 한다. 분석부분의 흐름은 (그림 4)와 같다.



(그림 4) 분석부분의 순차도



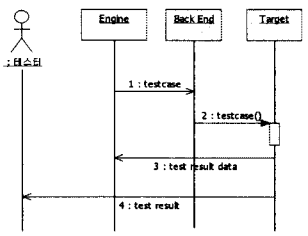
(그림 5) 연산부분의 순차도

연산부분은 분석데이터를 기반으로 테스트케이스를 생성하는 부분이다. (그림 5)는 연산부분의 순차도를 나타낸다. 분석데이터에서 테스트케이스를 생성할 때 사용되는 방법은 관측가능성 방법을 기반으로 한다. (그림 6)은 이 방법에 대한 예를 나타내는데, 이 방법은 입력값이 출력값까지 도달하면서 출력값에 영향을 미치지 못한 명령어를 관측하는 방법이다. 즉 출력값에 영향을 준 문장을 확인하여 모든 문장들이 결과값에 영향을 주는 값들로 테스트 값을 구성하는 것으로 모든 경로를 실행하는 방법, 모든 문장을 실행시키는 방법에 비해 적으면서도 모든 문장들을 테스트해볼 수 있다.

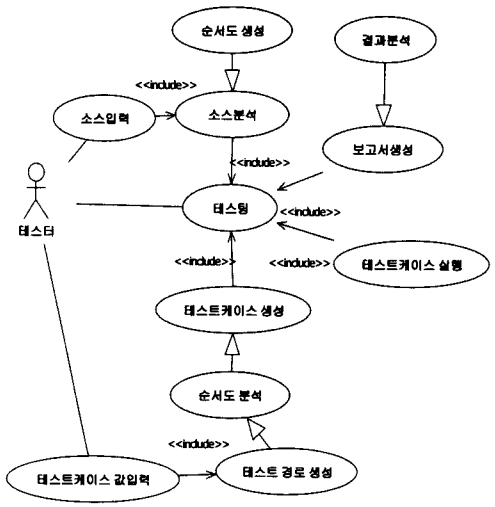
연산부분은 우선 호스트에서 먼저 흐름도를 실행하여 프로그램안의 모든 문장을 실행 후 출력값에 영향을 미칠 수 있는 최적의 테스트 케이스와, 출력

	a=1,b=1	a=-1,b=1	a=-1,b=-1
(1) Input a;	a → (1)	a → (1)	a → (1)
(2) Input b;	b → (2)	b → (2)	b → (2)
(3) x=a+1;	x → (1),(3)	x → (1),(3)	x → (1),(3)
If(a=b)			
(4) y=x+1;	y → (1),(3),(4)		y → (1),(3),(4)
else			
(5) y=a+b;		y → (1),(2),(5)	
If(a>0)			
(6) z=x;	z → (1),(3),(6)		
else			
(7) z=y;		z → (1),(2),(5),(7)	z → (1),(3),(4),(7)
(8) OUTPUT z;	Out → (1),(3),(6)	Out → (1),(2),(5),(7)	Out → (1),(3),(4),(7)

(그림 6) 관측가능성 방법의 예



(그림 7) Back End의 순차도



(그림 8) 테스팅 도구의 유스케이스 모델

값에 영향을 미치지 못하는 값들을 구분한다.

테스트케이스에 경우 하나의 테스트케이스가 타깃에서 실행할 수 있는 최소의 단위로서, 타깃에서 실행 후의 결과를 호스트로 전송하는 명령이 추가된다. 테스트케이스가 성공적으로 수행되었는가를 측정하는 것은 테스트케이스의 출력값이 원하는 결과를 발생시킬 수 있는가의 여부로서 판단할 수 있다[5].

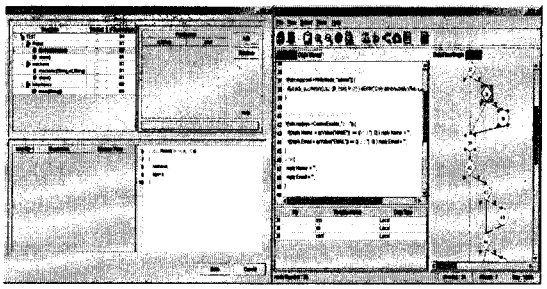
엔진은 사용된 테스트케이스를 저장하여 수정 후 재테스트시의 다시 활용이 가능하게 한다.

Back End인 연결부분은 호스트와 타깃을 연결하는 부분으로서, 다양한 연결방식에 맞추어 호스트와 타깃의 연동이 가능하도록 한다. (그림 7)은 Back End의 순차도를 나타낸다.

(그림 8)은 테스팅 도구의 유스케이스 모델로서 테스팅 요소들을 나열한 것이며 (그림 9)는 실행된 테스팅 도구를 나타내는 것으로 분석부분에서 소스가 분석된 것과 데이터의 흐름을 나타내고 있다.

5. 결론

초기 단순한 산업/군사 분야에서 사용되기 시작한 임베디드 시스템은 그 후 사용분야가 확장됨에 따라 그에 사용되는 응용 프로그램 또한 복잡해지기 시작했다. 임베디드 시스템은 보통 일반적으로 사용되는 프로그래밍 언어와 다르고 제한이 따르기 때문에 일반적인 소프트웨어 프로그래밍 방식으로 태스팅 하거나 기존에 사용된 간단한 테스팅으로 현재의 임베디드 소프트웨어들을 테스팅하는 것은 무리가 있다.



(그림 9) 테스팅도구의 실행 예

본 논문은 임베디드 소프트웨어 테스팅 툴을 설계하였다. 제안된 툴은 프로그램의 소스를 기반으로 하는 테스팅 프로그램으로 이 툴은 다양한 플랫폼의 임베디드 소프트웨어에 대한 테스팅이 가능하도록 선택적으로 프로그래밍 언어, 테스팅 방법등을 설정할 수 있도록 하여 추후 성능 향상, 유지보수 등이 용이할 것이다.

참고문헌

[1] J. Costa, S. Devadas, and J. Monteiro. "Observability analysis of embedded software for Coverage-Directed validation" In Proceedings of the International Conference on Computer Aided Design, pages 27-32, 2000.

[2] Larry J. Morell, Lionel E. Deimel "Unit Analysis and Testing" Curriculum Module (SEI-CM-9-2.0). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.

[3] 김상일, 이남용, 류성열 "실시간 이동형 내장 소프트웨어 시험 도구의 구조 설계" 한국정보과학회 논문지 2006.04

[4] Fredrik Olsson. Henrik Lundberg. "Automated Module Testing of Embedded Software Systems" Lund Institute of Technology 2003

[5] 김지혁, 김상일, 류성열 "타깃 기반의 임베디드 소프트웨어 테스팅 기술 및 시스템 개발에 관한 연구" 한국정보처리학회 2003년 추계학술대회