임베디드 소프트웨어 테스팅 도구를 위한 아키텍처에 관한 연구

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

A Study on Architecture for Embedded Software Testing Tool

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

요 약

임베디드 소프트웨어는 IT기술의 발달과 하드웨어의 다양한 보급 등으로 인해 널리 사용되기 시작했으며, 일반적인 소프트웨어와는 다른 방식으로 제작되며, 절대 표준이 없기 때문에 다양한 방식으로 개발되고 있다. 임베디드 소프트웨어 품질 및 생산성을 위해서는 체계적인 테스팅 방법론이 요구된다. 기존의 테스팅 방법으로는 복잡하고 높은 수준의 임베디드 소프트웨어 기능들을 테스트하는데 한계점이 많다. 본 논문에서는 기존 소프트웨어 테스팅과 임베디드 소프트웨어 테스팅의 차이점을 제시하고 효율적인 임베디드 소프트웨어 테스팅이 가능한 지원도구의 요구사항과 전형적인 방법을 탈피하여 자동화된 테스팅 방법 및 도구인 ESTE(Embedded Software Testing Environments)의 구조와 지원 기능을 제시하고자 한다.

1. 서론

최근 IT기술의 발달과 하드위에의 다양한 보급으로 인해 모든 정보 산업기기에 IT기술이 내장되는 “Embedded Everywhere”시대가 도래하고 있다. 기기들의 발전은 기존에 사용되던 기기를 거의 동일한 성능을 가진 상태에서 작은 크기로 개발되거나, 나아가 하나의 기기에 여러가지 기기들이 혼합되는 컨버전스가 이루어지고 있다. 이러한 시대에서 소프트웨어의 비중은 매우 높아지고 있으며, 이러한 소프트웨어 기술 중에서도 Post-PC 시대의 주요 산업으로 대두 되는 임베디드 소프트웨어 기술에 대한 관심이 높아지고 있다.

소프트웨어(특히 임베디드 소프트웨어)가 점차 복잡해짐에 따라 이를 테스팅하는 기술 또한 높은 수준을 필요로 하게되고, 임베디드 소프트웨어의 경우 일반적인 소프트웨어와는 개발 방식이나 목적이 다르기 때문에 테스팅이 일반적인 방식과는 다를 수 밖에 없으며, 다양해 질 수밖에 없다. 소프트웨어가 시장 적시성에 맞게 빠르게 개발되기 위해서는 하나의 소프트웨어를 위한 테스팅 보다는 다양한 시스템, OS등에 적용 가능해야 하고 복잡 다양한 소프트웨어 테스트를 위해 테스트 슈트도 상당한 크기가 요구되에 기존 임베디드 시스템들을 위한 단순 인터페이스이나 지원되는 하드웨어에 의존하여 실행되는 테스트로는 한계가 있다.

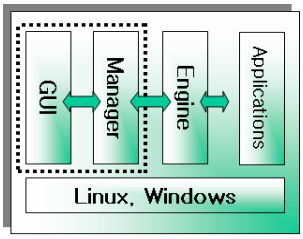
본 논문은 기존 소프트웨어 테스팅과 임베디드 소프트웨어 테스팅을 비교하고 임베디드 테스팅 도구의 요구사항과 구조, 지원기능을 위해 연구 중인 ESTE(Embedded Software Testing Environments)를 소개 한다.

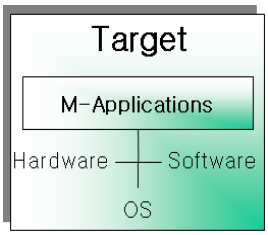
2. 관련연구

본 장에서는 기존에 사용되던 기존 소프트웨어의 테스팅과, 임베디드 소프트웨어 테스팅을 소개하고 그 차이점을 설명한다.

2.1 소프트웨어 테스팅

일반적인 소프트웨어 테스팅의 경우, 그림 1의 (a)





(a) 기존 소프트웨어 테스팅 구조 (b) 임베디드 시스템 테스팅 구조

(그림 1) 테스팅 구조

와 같은 구조를 가지며, 크게 테스트할 소프트웨어의 코드를 받아들이고 이를 분석하고 테스트를 수행하여 결과를 제공하는 구조로서 이루어진다.

임베디드 소프트웨어는 일반적인 소프트웨어와는 다른 방식으로 개발되는데, 일반적인 소프트웨어의 경우 소프트웨어만을 개발하는데 비해 사용 목적에 맞는 하드웨어와 함께 개발되는 경우가 많으며, 확고한 표준과 규격이 정해져 있지 않아 종류가 다양하다. 이는 그림 1의 (b)에서 나타나는데, 이로인해 존재하는 다양한 OS의 활용이 가능하며, 나아가 독자 개발할 수도 있다는 이점이 있다.

현재의 임베디드 시스템은 기존 임베디드 시스템보다 복잡해졌으며 디지털 가전기기, 음향, 센서, 모바일 기기등 분야가 다양해졌다. 이에 따른 요구사항들도 증가하여, 여러 가지 복잡한 작업을 수행할 수 있는 임베디드 시스템의 등장과 임베디드 소프트웨어의 발전으로 기존의 테스팅은 부족하거나 적합하지 않게 되었다.

이는 임베디드 시장이 증가하고 기기들이 발달함에 따라 소프트웨어도 복잡하게 되어, 기존 펌웨어 수준검사에 머물던 기존 소프트웨어 테스팅은 부적합하게 되었기 때문이다.[6]

2.2 임베디드 소프트웨어 테스팅 조건

임베디드 소프트웨어의 조건은 다음과 같다.[5]

① 실시간 테스팅(real time)

임베디드 소프트웨어는 실시간으로 제공되는 서비스를 기반으로 동작되므로 테스트는 사용자 환경과 동일하게 실시간으로 진행되어야 한다.

② 비간섭 테스팅(Non-intrusive)

수행중인 프로세서에 간섭을 주지 않은 상태에서 테스트가 수행되어야 한다. 즉, 테스트 하드웨어 동작에 영향을 미치지 않아야 한다.

③ 다양한 연결방식 지원

임베디드 시스템을 포팅(porting)된 펌웨어가 지원하는 연결 방법을 통해서만 외부와 통신을 수행할 수 있다. 따라서 각각의 펌웨어가 지원할 연결방식을 적절히 지원하기 위해 다양한 연결방식을 지원해야 한다.

3. 임베디드 소프트웨어 테스팅 도구의 구조

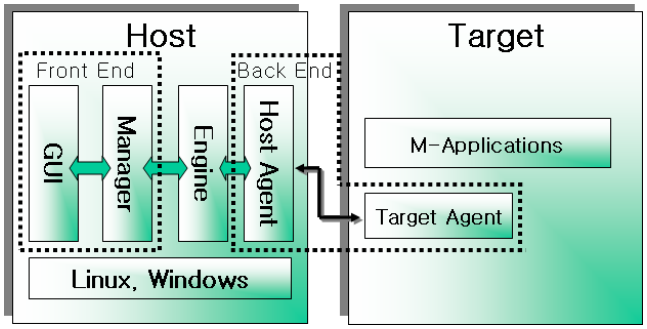
이 장에서는 임베디드 소프트웨어 테스팅 도구에 필요한 요구사항과 연구 중인 ESTE의 구조와 특징을 설명한다.

3.1 ESTE 요구사항

테스팅은 품질 보장에 대한 지원을 수행하는 중요한 역할을 가지는데, 크게 테스트 케이스의 설계, 소프트웨어와 테스트 케이스의 실행, 실행 결과 검토의 3가지 활동으로 구성된다. 임베디드 소프트웨어 테스팅은 하드웨어와 매우 밀접한 연관이 요구되며 임베디드 소프트웨어의 다양성처럼 테스팅 방식도 다양하며, 같은 결과에서 다양한 사람들이 각기 다른 요소를 원하는 경우도 있다. 테스트 영역은 시스템 구동 소프트웨어와 연결된 하드웨어(버튼, 스위치, USB등)로 나누어지며, 소프트웨어가 복잡해짐에 따라, 이에 따른 테스트 슈트의 크기도 증가되어 테스트할 소프트웨어와 비슷하거나, 커지는 경우를 위한 대비가 필요하다. 또한 반복적으로 테스트하는 것이 좋은데, 수정작업 후 수정의 영향으로 다른 결함의 발생하거나 또는 시스템에 영향을 줄 수 있기 때문이다. 이러한 반복적인 수행은 자동화하여 진행하는 것이 효율적이다.

3.2 ESTE 아키텍처

연구되는 임베디드 소프트웨어 테스팅 도구인 ESTE의 구조는 그림 2와 같으며, 임베디드 소프트웨어 테스팅의 요구사항을 고려하여 구성한다. 목표



(그림 2) ESTE 구조

로서 새로운 호스트나 타겟에 대한 호환성과 사용자가 시스템에 대한 상세한 지식 없이도 사용가능한 환경을 구축하는 것을 목표로 하고 있다. ESTE의 구조는 호스트와 테스트할 타겟으로 구분된다. Front End는 사용자가 입력한 명세, 값 등을 테스트 엔진에 전달하고, 테스트 엔진에서 테스트된 결과를 정리하여 사용자에게 제시한다. Back End의 경우는, 호스트와 타겟을 연결하는 구조로 되어 있다. 타겟에서 실행되는 어플리케이션을 분석하여 호스트로 보내며, 분석 내용을 토대로 테스트 사항을 타겟의 어플리케이션에 전달하여 직접 실행하고 실행 결과를 다시 호스트에 보내는 역할을 담당한다.

연구 중인 ESTE는 크게 도구의 핵심을 이루는 핵심구조인 테스팅 엔진과 에이전트, 그 외의 GUI와 Manager로 구성된다.[3]

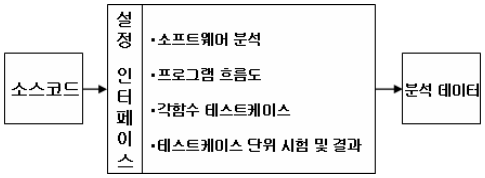
3.3 ESTE 기능

ESTE의 핵심구조로서 시스템의 중추라 할 수 있는 부분은 크게 두 가지로 나눌 수 있다. 이것은 테스트를 실행하는 엔진과 호스트와 타겟을 연결하는 Back End의 에이전트를 들 수 있다. 엔진의 경우 실제 테스트를 실행하고 결과를 표시하는 연산부분과, 연산 실행 전에 먼저 테스트할 임베디드 소프트웨어를 분석하는 분석부분으로 나눌 수 있다. 이러한 핵심구조는 독립적인 목적으로 사용할 수 있으며, 다양한 컴포넌트들과 연계하여 다양한 모습으로 나타낼 수 있다.

① 엔진 : 분석기(Analyzer)

테스팅 엔진은 크게 두개의 부분으로 나눌 수 있는데, 앞서 설명한 연산과 분석 부분이다.

분석부분의 구조는 그림 3과 같다. 분석부분의 경우 우선 테스팅을 실시하기 전에 테스트할 임베디드 소프트웨어를 분석하는 것이 주요 목적이다. 또



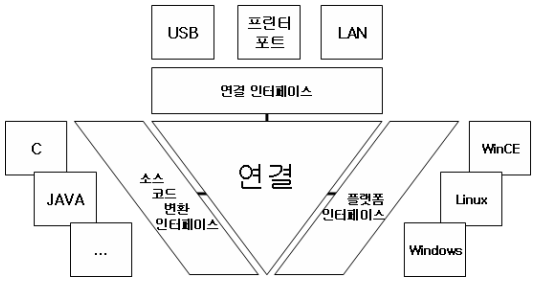
(그림 3) 분석 부분

한 기존에 분석된 내용을 저장하기도 한다. 분석이란 시스템의 이해라 생각할 수 있는데, 기능과 정보로서 분석한 소스코드를 표현하게 된다. 예로서 원시코드가 소스코드 구문 정보 추출에 의해 Token을 인식하고 패턴매칭으로 처리되어 매치된 패턴정보가 정보 추출기에 보내져 다시 구문정보로서 분석기로 보내진다. 또한 테스트할 임베디드 소프트웨어에서 구조를 분석하고 이 구조를 통하여 정해진 테스트 깊이의 수준을 작성 한다. 작성된 데이터는 연산 부분으로 전송되어 작업을 계속하게 된다.

② 엔진 : 실행기(Executer)

테스팅 엔진의 연산부분은 분석 부분에서 받은 데이터를 토대로 실제 테스트를 수행하여 테스트 결과를 출력하게 되며 테스트한 결과를 저장하고 다음 테스팅을 위한 피드백으로 사용할 수 있다. 연산 부분의 구조는 그림 4와 같다. 테스팅 엔진을 분석과 연산으로 나눔으로써, 실제 테스팅을 실행하게 되는 경우, 이미 제작된 테스팅 엔진 외에도 다른 테스트 엔진을 사용하거나, 한 분야의 임베디드 시스템을 위해 특화 또는, 여러 테스팅 엔진이 연합하는 등의 다양한 상황과 테스팅 사용자의 요구 목적에 맞는 정보 출력 작업 등이 용이하게 수행될 수 있다. 분석에서 얻어진 정보를 바탕으로 우선 정적 분석, 즉 소프트웨어 품질을 분석하는데, 크기, 구조 매트릭스를 통해 소프트웨어 품질평가를 수행하는 기능 메트릭스와 ISO/IEC 9126에 정의된 유지보수 품질 측정 및 평가로 수행하는 유지보수 품질 매트릭스로 이루어진다. 정적 분석기에서 매트릭스를 코딩규칙과 매트릭스 데이터베이스에서 정의하여 분석하고, 평가값을 계산하여 다시 데이터베이스에서 분석한다. 정적분석 후의 과정은 유닛 테스트로서 테스트케이스를 설계

(그림 4) 연산 부분



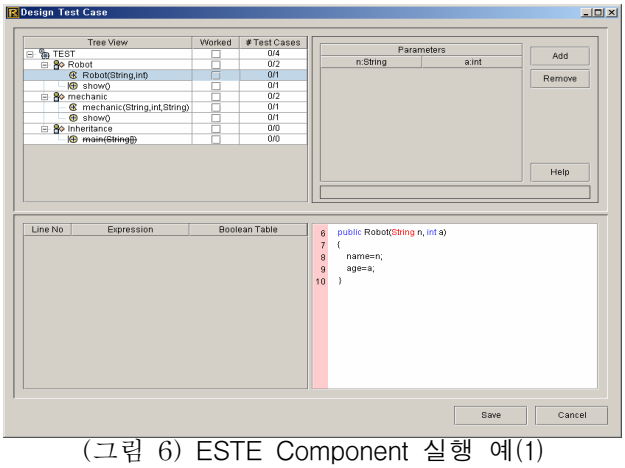
(그림 5) 연결 부분

하여 테스팅을 수행하는 과정이다. 컨트롤데이터의 흐름과 정적 커버리지를 테스팅하는 과정은 코딩규칙과 매트릭스 데이터베이스에서 가져온 코드와 테스트케이스를 가지고 테스팅을 실행하게 된다. 테스팅 엔진의 경우 보통 분석과 연산을 같이 수행하게 되지만, 분석과 연산을 분리하여 분석 부분에서 분석한 데이터를 여러 연산 방법으로 연산할 수 있으며, 유사하거나 비슷한 장비로의 응용에도 사용할 수 있다.[1,2,4]

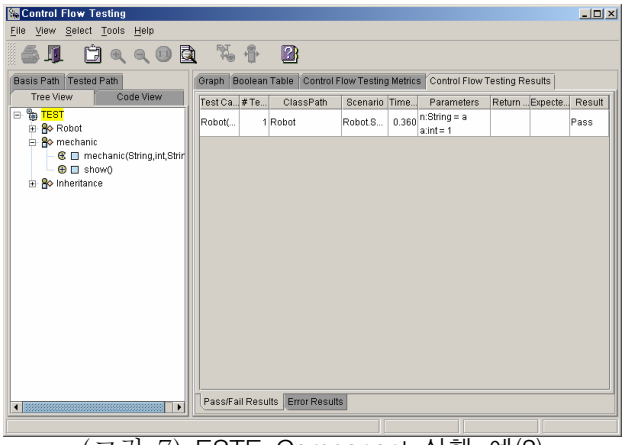
③ 에이전트 : 연결부(Connecter)

연결 부분은 호스트와 타겟을 연결하는 것 외에도, 호환되지 않는 시스템(윈도우와 리눅스 등)과 지원되는 드라이버 등의 요소들을 포함할 수 있다. 연결 부분의 구조는 그림 5와 같다. 타겟에 필요한 드라이버가 호스트의 시스템에서 작동하지 않는 경우에, 이를 호스트용으로 다시 작성하거나 포팅 등의 작업을 거쳐야 하지만, Back end의 에이전트를 이용하여 서로 간의 연결과 드라이버의 사용이 가능하도록 하게 할 수 있다. 에이전트의 장점은 하나의 테스팅 도구로서 독립적으로 작동하는 것 외에도 다른 컴포넌트 등에 탑재되어 작동될 수 있다는 것이다.

4. ESTE Component 실행 예



(그림 6) ESTE Component 실행 예(1)



(그림 7) ESTE Component 실행 예(2)

5. 결론

테스팅의 중요성과 기존 소프트웨어와 임베디드 소프트웨어 테스팅의 차이점, 그리고 최종적으로 임베디드 소프트웨어 테스팅 도구의 구조와 지원기능에 대해 본 논문에서 서술하였다. 테스팅 도구 사용에 의한 효과로서 소프트웨어의 질을 높일 수 있으며, 임베디드 소프트웨어를 테스트하는데 드는 시간과 비용을 절감 시킬 수 있다. ESTE는 제안된 핵심 구조를 우선적으로 개발 중이며, 기술적인 관점에서는 이 핵심 구조들이 다양한 컴포넌트와의 결합으로 여러 분야에서 사용 가능하다. 이것은 도구의 업그레이드나 변경이 용이하여 유지보수의 관점에서도 상당한 효과를 거둘 수 있을 것이다.

참고문헌

[1] 이용호, 정창신, 신석규, "임베디드 소프트웨어 시험 사례", 정보과학회지 제22권 제6호, pp68-76, 2004.

[2] 양해술, 이하용, "임베디드 소프트웨어 품질 평가 모델의 개발", 한국정보처리학회 추계 학술발표논문집 제9권 제2호, pp1-4, 2002.

[3] 최현미, 성아영, 조나경, 최병주, 반효경, 김재웅, "임베디드 소프트웨어 평가 모델 구축 방안" 한국정보과학회 가을 학술발표논문집 Vol.31, No.2, 2004

[4] Mary Jean Harrold, "Testing: A Roadmap" 22nd International Conference on Software Engineering, 2000

[5] Techniques and Tools for Software Analysis, Technical paper, METROWERKS http://www.metrowerks.com/

[6] Harro S. Jacobs, Peter H.N. "Embedded TestFrame, An Architecture for Automated Testing of Embedded Software" Workshop on Embedded systems (PROGRESS2000), pp. 47-52, 2000