KIPS Tr. Software and Data Eng.
Vol.6, No.5 pp.245~256 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2017.6.5.245

A Hierarchical Checklist to Automatically Generate Test Scripts

Dae Joon Kim[†] · Ki Hyun Chung[††] · Kyung Hee Choi[†††]

ABSTRACT

This paper proposes a method to generate test scripts for testing embedded system in an easy manner by using hierarchical checklist. In the proposed method, a checklist is constructed with event, component and command dictionaries. And the test scripts are hierarchically generated based on the dictionaries. Since the physical layer of system input becomes abstract with component layer and event layer by virtue of the hierarchy, It is possible to generate test scripts without complicated system input information. It is easy to generate test scripts for embedded systems with similar inputs using the highly reusable dictionaries. The effectiveness of the proposed method is demonstrated with experiments.

Keywords : Embedded System, Test, Test Script, Checklist

테스트 스크립트 자동 생성을 위한 계층 구조 체크리스트

김 대 준[†] · 정 기 현[††] · 최 경 희[†††]

요 약

본 논문은 구조화된 체크리스트로부터 임베디드 시스템 테스팅을 위한 테스트 스크립트를 쉽게 생성할 수 있는 방법을 제안한다. 제안하는 방법은 체크리스트를 이벤트(Event), 컴포넌트(Component), 입력 명령어(Command) 사전을 기반으로 구성하고, 사전으로부터 계층적으로 테스트 스크립트를 생성한다. 계층 구조로 임베디드 시스템의 물리적 입력 계층이 상위 계층의 컴포넌트 및 이벤트 계층에서 추상화되어 복잡한 시스템 입력 정보를 사용하지 않고도 테스트 스크립트를 생성할 수 있다. 비슷한 종류의 입출력 정보를 가지는 임베디드 시스템을 테스트하기 위한 테스트 스크립트 생성은 재사용성이 높은 사전을 이용하여 매우 쉽게 할 수 있다. 제안하는 방법의 유용성은 실험을 통해 보인다.

키워드 : 임베디드 시스템, 테스트, 테스트 스크립트, 체크리스트

1. 서 론

오늘날 임베디드 시스템은 모바일 기기, 가전제품, 의료기기, 자동차, 항공기, 군사용 무기 등 다양한 분야에서 활용되고 있다. 과거에 비해 임베디드 시스템의 기능은 점점 더 복잡해지고 많은 요구사항을 처리하고 있으며, 이에 따라 임베디드 시스템의 기능을 수행하는 소프트웨어 또한 복잡해지고 있다.

임베디드 시스템에서 발생하는 오류의 많은 부분은 소프트웨어적인 오류이며[1], 의료기기, 자동차, 항공기, 군사용 무기 등 신뢰성이 매우 중요한 시스템에서 오류 발생 시 심각한 문제를 초래할 것이다. 이처럼 신뢰성이 매우 중요한 시스템은 오류 발생을 예방하기 위해 철저한 테스트가 필요하다.

임베디드 시스템을 테스트하기 위한 방법은 크게 블랙박스(Black Box)와 화이트박스(White Box)가 있다. 화이트박스는 내부 소스코드를 분석하여 소스코드가 설계 사양, 조건, 코딩 규칙 등에 대하여 올바르게 작성되었는지를 확인하는 방법이고, 블랙박스는 내부 소스코드를 고려하지 않고 시스템 기능이 정해진 요구사항대로 정확하게 동작하는지를 확인하는 방법이다.

블랙박스 테스트를 하기 위한 다양한 방법 중 산업 현장에서는 엔지니어의 경험을 기반으로 하는 테스트 체크리스트(Checklist)를 많이 사용하고 있다. 체크리스트란 테스트하고자 하는 목록을 작성한 문서이며, 체크리스트 기반 테스트는 체크리스트에 작성된 목록에 따라 시스템이 올바르게 동작하는지를 확인하는 방법이다.

요구사항을 기반으로 작성된 체크리스트를 이용한 블랙박스 테스트는 화이트박스에서 검출하기 어려운 오류 검출이 가능하다. 예를 들어 온도 제어 장치 요구사항에 온도가 20

※ 본 연구는 방위사업청(UD150042AD)의 지원으로 수행되었음.
† 준 회 원 : 아주대학교 컴퓨터공학과 석사과정
†† 정 회 원 : 아주대학교 전자공학과 교수
††† 정 회 원 : 아주대학교 컴퓨터공학과 교수
Manuscript Received : October 17, 2016
Accepted : December 4, 2016
* Corresponding Author : Ki Hyun Chung(khchung@ajou.ac.kr)

도 이하일 경우 온도 조절 장치를 동작해야 한다고 작성되어 있다면, 소프트웨어 엔지니어는 요구사항을 확인하고 온도 20도 이하 동작에 맞는 소스 코드를 작성한다. 만약 소프트웨어 엔지니어의 실수에 의해 온도 10도 이하로 소스 코드를 작성하였다면, 이 코드는 요구사항과 불일치하는 오류의 코드이다. 하지만 화이트박스를 이용한 정적, 동적 테스트는 소스 코드의 변수, 코딩 규칙, 조건이 만족하기 때문에 옳은 코드로 판단한다.

본 논문은 체크리스트로부터 다양한 테스트 케이스를 생성하고 임베디드 시스템을 테스트할 수 있는 물리적 입력 값인 테스트 스크립트를 체계적, 계층적으로 생성할 수 있는 방법을 제시한다.

일반적으로 테스트 엔지니어는 체크리스트에 작성된 내용을 참조하여 실제 테스트 대상 시스템(SUT: System Under Test)에 인가되는 시스템 입력 값을 수작업으로 도출한다. 이러한 방법은 몇 가지 단점이 있다.

첫째, 체크리스트에 작성된 내용은 인간이 이해할 수 있는 자연어에 가깝게 작성되어 있고, 시스템 입력 값은 시스템이 이해할 수 있는 기계언어에 가깝다. 이 형식의 큰 차이를 극복하고 자연어로 작성된 체크리스트를 참조하여 시스템 입력에 필요한 기계어로 만들기 위해서는 엔지니어의 많은 노력이 필요하다.

둘째, 테스트 엔지니어는 체크리스트에 작성된 내용만을 중점적으로 테스트한다. 이 의미는 체크리스트에 작성된 내용은 테스트가 가능 하지만 (체크리스트에 작성되지 않은) 그 외의 오류 발생 가능성이 있는 동작은 테스트하기 어렵다는 의미이다. 예를 들어 체크리스트 항목에 “온도가 20도 이상일 경우 특정 센서를 동작시켜야 한다.”는 내용이 작성되었다면, 실제 체크리스트 항목에 작성된 “온도 20도 이상” 뿐만 아니라 오류 발생 가능성이 있는 “온도 20도 미만” 까지도 테스트할 수 있어야 한다.

셋째, 체크리스트에 작성된 내용을 테스트 엔지니어가 수작업으로 분석하는 과정에서 엔지니어의 주관적인 생각 또는 실수에 따라 테스트 결과가 달라질 수 있다. 또한 테스트 항목이 자주 바뀌거나 추가된다면, 추가된 항목의 조건들을 고려하여 수작업으로 시스템 입력 값을 다시 생성해야 하기 때문에 많은 시간이 소비된다.

넷째, 임베디드 시스템 하드웨어의 정보를 정확히 모르는 경우는 기존의 체크리스트를 이용하여 테스트 스크립트를 만들기 힘들다.

본 논문에서는 이러한 단점을 해결하기 위해 체크리스트를 효율적으로 구조화 하는 방법과 구조화된 체크리스트로부터 체계적으로 테스트 스크립트를 생성하는 방법을 제안한다. 제안하는 방법은 체크리스트 유닛을 체계적으로 구조화하고, 테스트 스크립트를 생성하기 위해 체크리스트에서 참조하는 명령어 사전(Command Dictionary) 데이터를 작성하는 방법을 제안한다. 본 논문에서 제안하는 방법에서는 체크리스트를 활용한 이전 연구[29]의 불편한 점을 개선하기 위하여 계층 구조를 활용하여 테스트 스크립트 도출은 더 쉽게 할 수 있고 스크립트 생성에 사용하는 명령어의 재사용성을 높일 수 있다. 제안된 방법에서는 테스트 스크립트를 체계적으로 도출하기 위해 체크리스트를 크게 이벤트(Event), 컴포넌트(Component), 입력 명령어(Command), 시스템 입력(System Input)의 계층으로 나누어 작성한다. 제안한 방법을 사용할 경우, 하드웨어 동작에 대한 자세한 정보를 가진 엔지니어가 작성해 놓은 명령어 사전과 컴포넌트 사전을 활용하면 하드웨어 정보와 무관하게 테스트 스크립트를 작성할 수 있다. 즉, 인간이 이해하기 쉽고 하드웨어 계층이 추상화된 이벤트 명령어로 구성된 체크리스트로부터 테스트 시 시스템에 인가되는 입력 데이터인 테스트 스크립트를 쉽게 도출하는 것이 가능하다.

본 논문의 구성은 다음과 같다. 제 2장은 관련연구를 소개하고, 제 3장은 체크리스트 체계와 테스트 스크립트 생성 방법을 제안한다. 제 4장은 제안하는 방법을 바탕으로 실험을 진행하고, 그 결과로 제안하는 방법의 효과를 보인다. 마지막으로 제 5장은 본 논문의 결론을 기술한다.

2. 관련 연구

SUT를 테스트하기 위한 테스트 케이스 생성 방법으로는 크게 화이트 박스(White Box)와 블랙박스(Black Box) 테스트 케이스 생성 방법이 있다. 화이트 박스 생성 방법은 소스코드를 분석하여 코드의 문법, 코드의 규칙 및 코드의 실행 오류(Run Time Error) 등을 검출하는 테스트 케이스를 생성하는 방법이다.

블랙박스 테스트 케이스 생성 방법은 시스템 내부 소스코드를 고려하지 않고 완성된 시스템을 대상으로 요구사항대로 기능이 정확히 동작하는지 또는 특정 시나리오에서 오류가 발생하는지 등을 검출하기 위한 테스트 케이스 생성 방법이다. 블랙박스 테스트는 목표에 부합되는 입력 생성의 어려움은 있으나, 소스코드의 확보나 시스템의 구현 상세 정보 없이 테스트 케이스를 만들 수 있다는 장점이 있다.

블랙박스 테스트 케이스 생성 방법의 종류로는 무작위(Random) 생성 방법, 시스템 입력 조합(Combinatorial)을 이용한 생성 방법 및 모델 기반(Model-based) 생성 방법[17] 등이 있다.

앞서 설명한 구조적인 테스트 케이스 생성 방법과 더불어 실제 산업 현장에서는 엔지니어의 경험을 기반으로 테스트하는 경험 기반 테스트(Experience-based technique)방법 또한 많이 사용하고 있다. 경험 기반 테스트 케이스 생성 방법은 테스트 엔지니어의 과거 경험이나, 사용자의 피드백 등의 오류 데이터베이스를 활용하여 테스트 케이스를 생성한다.

경험 기반으로 테스트 케이스를 생성하기 위해 실제 산업 현장에서는 체크리스트를 많이 사용하고 있다. 체크리스트는 테스트하고자 하는 내용 또는 시나리오로 작성되고, 테스트는 체크리스트에 기반 하여 SUT를 테스트한다.

우리가 아는 지식으로는 아직까지 체크리스트를 활용하여

체계적으로 테스트 스크립트를 생성하는 방법은 많지 않다. [29]에서는 체크리스트를 구조화하기 위해 [23]의 저자들이 설명한 유스 케이스(Use case) 아이디어를 활용한다. 유스 케이스에서 사용하는 유스 케이스 명세서 구성은 크게 시나리오 흐름과 조건으로 나누어진다. 시나리오 흐름은 시스템이 사용자의 요구 기능을 정상적으로 수행하는 기본 흐름(Basic Flow)과 요구 기능의 수행도중 실패할 경우 적절히 대체해야 할 대체흐름(Alternative flow)으로 구성된다. 시나리오 조건은 유스 케이스의 수행이 시작되기 위한 선행조건(Precondition)과 수행이 완료된 후에 만족되어야 하는 후행조건(Postcondition)으로 구성된다.

[29]에서는 일반적으로 사용하는 체크리스트의 단점인 체크리스트로부터 다양한 테스트 케이스를 체계적으로 생성하기 힘들다는 점과 테스트 스크립트를 생성하기가 힘들다는 점 등을 해결하고자 체크리스트로부터 테스트 스크립트를 자동으로 생성하는 방법을 제안한다. 체크리스트에서 테스트 스크립트로 변환하기 위해 필요한 정보가 작성된 테스트 명령어 사전(Test Command Dictionary)을 정의하고 정의된 명령어를 사용하여 테스트 스크립트를 도출하는 방법을 제안한다. Fig. 1은 [29]에서 제안하는 테스트 명령어 사전의 예이다.

명령어	group	TIME	SysIn_DSensor					
D<50	1	0	49					
명령어	group	TIME	SysIn_DSensor					
D>=50	1	0	51					
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
정상	8	0	49	-10	15	0	0	
		5						0
명령어	group	TIME	SysIn_DSensor	SysIn_FSensor	SysIn_R1Sensor	FB_FFan	FB_CFan	SysIn_JetFreezing
D-ERR	8	0	-50	-10	15	0	0	
		5						0

Fig. 1. A Test Command Example in [29]

[29]에서 제안하는 테스트 명령어 사전은 시스템 입력에 인가되는 시스템 입력 값과 명령을 정의하고 있다. 예를 들면, Fig. 1에서 명령 “D<50”은 시스템 입력 “SysIn_DSensor” 입력의 “49” 값을 의미한다.

동작모드	기능분류	기능설명	테스트 명령
초기모드	진입시조건	D센서가 50 미만	D<50
		주 에러 없을 시	정상
	주동작	데모모드 동작	DemoMode 진입
	종료조건	30초 작업 후 정상 종료	30초
		D센서가 50 이상	D>=50
		D_Sensor 에러 발생 시 초기모드 종료	D-ERR
		F_Sensor 에러 발생 시 초기모드 종료	F-ERR
		R1_Sensor 에러 발생 시 초기모드 종료	R1-ERR

Fig. 2. A Checklist Example in [29]

Fig. 2는 [29]에서 제안하는 구조를 가진 체크리스트의 예이다. 체크리스트에 작성된 “테스트 명령” 열은 테스트 하고자하는 명령어를 작성하는 공간이며, 이 명령어는 Fig. 1 테스트 명령어 사전에 정의된다. 예를 들어 체크리스트의 “테스트 명령”에 작성된 “D<50” 은 D센서가 50도 미만인지 테스트 하는 조건이다. “D<50” 이라는 테스트 명령을 실행함으로써 Fig. 1에 작성된 “D<50” 명령어의 “SysIn_DSensor” 입력 값 “49”를 도출할 수 있다.

하지만 [29]에서 제안하는 방법은 몇 가지 개선점이 필요하다. 첫째, 명령어 사전은 하나의 단순한 명령어 구조를 사용하고 있어서 명령어 재사용이 어렵다. 둘째, 테스트 케이스의 실제 물리적 입력인 테스트 스크립트는 시스템의 정보를 정확히 알 수 있어야 작성이 가능하다. 때문에 스크립트 생성 시 마다 하드웨어 정보를 자세히 참조해야 하므로 스크립트 작성이 어렵다. 이러한 단점을 완화하기 위해 본 논문에서는 명령어 체계에 계층적 구조를 도입하여 이전 방법의 단점을 획기적으로 개선하고자 한다.

3. 체크리스트 기반 테스트스크립트 생성

3.1 체크리스트 구조화 방법

제안하는 체크리스트 구조는 이벤트(Event), 컴포넌트(Component), 입력 명령어(Command) 및 시스템 입력(System Input) 4단계로 분류하는 방법을 제안한다. Fig. 3은 체크리스트 구조를 4계층으로 표현한 그림이다.

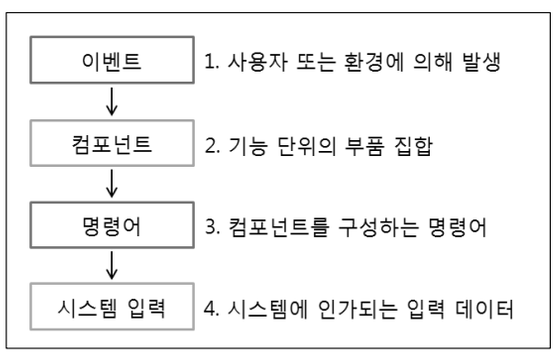


Fig. 3. Four Layers of the Checklist

첫 번째, 이벤트 계층은 환경 또는 사용자가 시스템이 반응하도록 조작하는 행위를 나타낸다. 예를 들면, 사용자가 시스템을 버튼 등으로 조작하거나 혹은 특정 환경(온도, 바람, 고도) 등에 의하여 시스템 동작이 바뀌는 것을 의미한다. 두 번째, 컴포넌트 계층은 기능 단위의 부품 집합을 표현한 단계이며, 시스템을 구성하는 센서, 모터 등의 부품을 나타낼 수 있다. 또한 여러 작은 컴포넌트를 구성하여 하나의 큰 컴포넌트를 표현할 수 있다. 세 번째, 명령어 계층은 시스템 동작이나 상태를 변화시키는 명령어를 표현한 계층이며, 시스템을 켜고 끄는 명령어는 PowerOn, PowerOff와 같이 정의할 수 있다. 시스템 입력은 해당 명령어를 수행하기 위한 실제 시스템에 인가되는 시스템의 실제 물리적 입력 데이터로 구성된다.

4계층을 간단히 자동차로 예를 들어 설명하면, 이벤트 계층은 사용자가 자동차의 기능을 조작하는 행위로 자동차 기어(Gear)를 1단, 2단, 또는 3단 등으로 변경하는 행위로 정의할 수 있고, 컴포넌트 계층은 자동차의 기능을 수행하는 부품 집합으로 기어변속장치(Transmission), 브레이크장치

(Break), 가속장치(Accelerator), 조향장치(Steering) 등으로 구성이 가능하다. 명령어는 시스템의 상태나 동작을 변화시키는 명령으로 기어변속장치를 켜고 끄는 명령어는 TransOn, TransOff 와 같이 정의할 수 있다. 마지막으로 시스템 입력은 TransOn 과 TransOff 명령에 의해 영향을 받는 시스템 입력 값을 의미한다. 예를 들어, 기어변속장치의 전원을 조작하는 시스템 입력의 변수 명이 SysIn_TransPower라고 정의되었다면, TransOn, TransOff 명령을 수행함으로써 SysIn_TransPower 입력으로 1 또는 0 이 인가될 수 있다.

Fig. 4는 체크리스트 계층구조를 나타낸 그림으로 이벤트는 여러 컴포넌트에 의해 구성되고, 컴포넌트는 여러 하위 컴포넌트 또는 명령어에 의해 구성된다. 또한 명령어는 여러 시스템입력으로 구성된다.

그림의 계층 구조를 살펴보면 이벤트와 컴포넌트는 밀접한 관계가 있고, 컴포넌트와 명령어는 밀접한 관계가 있다. 또한 명령어는 시스템 입력과 밀접한 관계가 있는 것을 볼 수 있다. 이러한 단계를 거침으로써 사용자 또는 사용 환경에 의해 발생되는 이벤트만을 사용하여도 시스템 입력 정보 없이 테스트 스크립트를 도출할 수 있다.

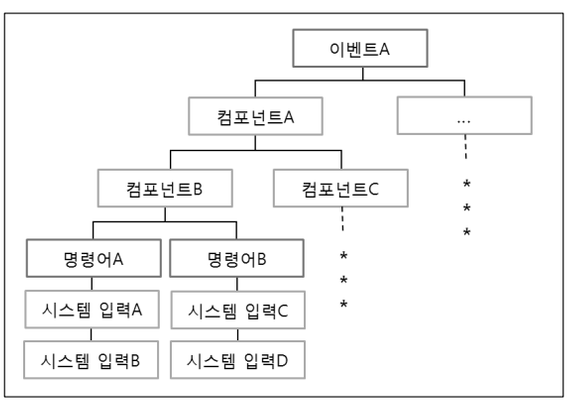


Fig. 4. A Hierarchical Checklist Example

3.2 테스트 명령어 사전

본 절은 “3.1 체크리스트 구조화 방법”에서 설명한 내용을 기반으로 체크리스트 구성에 필요한 테스트 명령어 사전(Test Command Dictionary) 파일을 작성하는 방법을 제안한다. 테스트 명령어 사전은 이벤트, 컴포넌트, 입력 명령어 사전으로 구성되며 텍스트(Text) 형식이나 엑셀(Excel) 파일 등의 형식으로 작성할 수 있다.

3.2.1 이벤트 사전

Fig. 5는 이벤트 사전 구조이다. 이벤트 테이블은 “이벤트”, “Group”, “컴포넌트 참조” 3가지 열로 구성된다.

“이벤트” 열은 사용자 정의 이벤트 이름을 정의하는 공간이다. “컴포넌트 참조” 열은 “3.2.2 컴포넌트 사전” 절에서 정의되는 컴포넌트를 사용하기 위해 작성하는 공간이다. “이벤트”와 “컴포넌트 참조” 열의 구조 설명은 “3.2.2 컴포넌트 사전” 절에서 자세히 설명한다.

“Group” 열은 동일하거나 상반되는 특성을 갖는 이벤트를 묶어주는 그룹 번호를 작성하는 열이다. 예를 들어, 시스템 전원을 켜고 끄는 이벤트를 “power on”과 “power off”이라고 정의하고, 이 이벤트들을 같은 그룹으로 설정한다면 “power on” 이벤트가 TRUE 조건이라고 가정했을 때 FALSE 조건으로 “power off” 이벤트가 선택될 수 있다. Fig. 5에서 이벤트A와 이벤트B “Group”의 값이 1로 정의되어 있다. 이 의미는 두 이벤트를 같은 그룹으로 설정하겠다는 의미이다.

파일타입:	이벤트

이벤트	group	컴포넌트 참조
이벤트A	1	컴포넌트A(10)
		컴포넌트B(20)

이벤트	group	컴포넌트 참조
이벤트B	1	컴포넌트C(30)
		TIME(200)
		컴포넌트D(40)

Fig. 5. An Event Dictionary Example

3.2.2 컴포넌트 사전

Fig. 6은 컴포넌트 사전 구조이다. 컴포넌트 테이블은 “컴포넌트”, “명령어 참조” 2가지 열로 구성된다. “컴포넌트” 열은 사용자 정의 컴포넌트 이름을 정의하는 공간으로 컴포넌트 이름과 매개변수(Parameter)로 구분된다. 매개변수는 “컴포넌트” 열에 작성된 “#Param”을 뜻하며, 매개변수 사용법은 “3.2.3 입력 명령어 사전” 절에서 자세히 설명한다.

파일타입:	컴포넌트

컴포넌트	명령어 참조
컴포넌트A(#param)	명령어A(#param)
	명령어B(100)

컴포넌트	명령어 참조
컴포넌트B(#param)	명령어C(200)
	컴포넌트K(100)

컴포넌트	명령어 참조
컴포넌트K(#param)	명령어E(300)
	명령어F(#param)

Fig. 6. A Component Dictionary Example

“명령어 참조” 열은 Fig. 7 입력 명령어 사전에 정의된 명령어를 사용하기 위해 작성하는 공간이다. “컴포넌트A(#param)” 테이블에 작성된 “명령어A(#param)”의 의미는 “컴포넌트A”에서 입력 받은 매개변수 “#param”을 “명령어A”의 매개변수로 전달하겠다는 의미이다. 마찬가지로 “명령어B(100)”는 “명령어B”의 매개변수로 100 값을 전달하겠다는 의미이다.

“컴포넌트A”를 발생시킴으로써 “명령어A”와 “명령어B”가 순차적으로 수행되고, 입력 명령어 사전에 정의된 “명령어A”와 “명령어B”를 실행시키고, 그 명령어들은 3.2.3의 입력 명령어 사전에 정의된 시스템 입력 값이 테스트 스크립트로 생성된다.

컴포넌트 정의는 여러 하위 컴포넌트들을 구성하여 하나의 큰 컴포넌트로 정의할 수 있는 계층적 구조를 가질 수 있다. 예로 “컴포넌트B(#param)” 의 명령어 참조에서는 “컴포넌트K(100)”을 정의했다. 이 의미는 “컴포넌트B”에서 하위 “컴포넌트K”를 참조하겠다는 의미이다.

3.2.3 입력 명령어 사전

Fig. 7은 입력 명령어 사전 구조로 SUT 테스트에 사용되는 명령어가 정의된다. 각 명령어는 SUT의 시스템 입력 중에서 해당 명령어 수행에 사용되는 입력들의 각 시간대 별 물리적인 값들이 정의된다.

파일타입:	명령어

명령어	TIME
TIME(#t)	#t

명령어	TIME	시스템 입력
		SysIn_SensorA
명령어A(#p)	10	#p

명령어	TIME	시스템 입력
		SysIn_SensorB
명령어B(#g)	0	50
	100	#g

명령어	TIME	시스템 입력	
		SysIn_SensorC	SysIn_SensorD
명령어C(#l, #r)	0	-1	-1
	10	#l	#r

명령어	TIME	SysIn_SensorE
명령어D(#a)	0	-50
	15	1000 * #a

명령어	TIME	SysIn_SensorF
명령어E(#s)	0	5000 + #s

Fig. 7. An Input Command Dictionary Example

Fig. 7에 작성된 명령어 테이블 구성은 “명령어”, “TIME”, “시스템 입력” 3가지 열로 구성된다. “명령어” 열은 사용자 정의 명령의 이름을 정의하는 공간으로 명령어 이름과 매개변수(Parameter)로 구분된다. 매개변수는 특수문자 “#” 으로 시작하며, 복수개의 입력을 선언할 수 있다. 매개변수는 “,” 로 구분된다.

예를 들어, “명령어” 열에 작성된 “명령어A(#p)” 의미는 명령 이름을 명령어A 라고 정의하고, #p라는 이름의 매개변수를 하나 입력받을 수 있다는 의미이다.

“시스템 입력” 열은 SUT에 인가되는 입력 데이터를 정의하는 공간으로 소수, 정수, 참/거짓 또는 매개변수를 입력받을 수 있다. “시스템 입력”은 하나의 “명령어”에 복수개의 정의가 가능하다. 여기서는 SUT의 모든 입력 값을 다 정의하는 것이 아니라, 해당 명령어 수행에 필요한 명령어만을 정의한다. 명령어에 정의되지 않은 입력 신호는 이전 값을 유지한다. (모든 시스템 초기 입력 값은 사전에 정의한다.)

“TIME” 열은 입력을 인가해야하는 시간을 기술한 공간으로, 각 입력 행간의 시간 간격은 입력이 가해지는 상대시간이다. 또한 “TIME” 열은 입력 받은 매개변수에 따라 시간을 변화 시킬 수 있다.

Fig. 7에 작성된 “명령어D(#a)”는 해당 명령어를 참조하는 컴포넌트 그리고 그 컴포넌트를 참조하는 이벤트 명령어에서 정의된 변수 값을 승계 받는다. Fig. 7 명령어 사전에 정의된 “명령어D(#a)”의 시스템 입력으로 “SysIn_SensorE” 가 정의되어 있다. 이벤트 및 컴포넌트가 “명령어D(10)”을 참조하였다면 가장 처음 “SysIn_SensorE” 입력으로 “-50” 값이 인가되고, 15ms 이후에 “1000 * #a” 값이 인가된다. 앞서 “#a”는 10으로 치환되므로 “1000 * 10 = 10000” 값이 “SysIn_SensorE”에 인가된다.

각 “시스템 입력”과 “TIME”에는 곱하기(*), 나누기(/), 빼기(-), 더하기(+) 등의 산술연산 사용이 가능하고, 괄호를 이용하여 사칙연산 우선순위 변경이 가능하다.

3.3 테스트 스크립트 자동 생성을 위한 체크리스트

테스트 엔지니어는 체크리스트에 작성된 항목을 이용하여 시스템 동작을 테스트한다. 산업현장에서 일반적으로 사용하는 체크리스트는 통일된 양식이 없고 목적에 맞게 양식을 만들어 사용한다. 본 절에서는 테스트 스크립트를 자동으로 생성하기 위해 사용하는 체크리스트는 [29] 연구에서 제안한 구조를 활용하며, Fig. 8과 같다.

체크리스트 구성은 “모드”, “기능설명”, “조건”, “이벤트” 4가지 열로 구성된다. “기능설명” 항목은 실제 테스트 스크립트 생성과 무관하며, 사용자에게 테스트 정보를 알려주는 용도로만 사용된다.

모드	기능설명	조건	이벤트
일반모드	선행 조건 이벤트A 를 동작시킨다.	Pre	이벤트A
	주 동작 이벤트B 를 동작시킨다.	Act	이벤트B
	주 동작 이벤트C 를 동작시킨다.		이벤트C
	주 동작 이벤트D 를 동작시킨다.		이벤트D
	종료 조건 이벤트E 를 동작시킨다.	Post	이벤트E
	종료 조건 이벤트F 를 동작시킨다.		이벤트F

Fig. 8. A Checklist Example

“모드”는 검사하고자 하는 모드 명을 기재하는 항목이다. 초기 모드, 일반 모드, 대기 모드 등과 같이 정의할 수 있으며, 고유의 기능 전체를 하나의 독립 적인 동작 그룹이 분류 될 수 있는 동작 모드 이름이다. 따라서 다른 분류의 조합으로 SUT의 다른 동작을 테스트할 수 있다.

“조건”은 각 모드 내에서 수행하는 기능을 말하며, “Pre”, “Act”, “Post” 항목으로 나누어진다. “Pre”는 해당 모드에

진입하기 위하여 만족해야 하는 조건이다. “Pre” 조건이 일부 혹은 모두 만족해야 “Act” 로 진입할 수 있다. “Act”는 해당모드에서의 동작 조건을 정의한다. 마지막으로 “Post”는 해당 모드를 종료하기 위한 정상 종료나 비정상 종료의 조건을 의미한다. “Post”는 “Pre”와 마찬가지로 조건이 일부 혹은 모두 만족할 경우 해당 모드를 종료한다. 각 모드의 수행 흐름은 “Pre → Act → Post” 순서로 수행된다.

“이벤트”는 테스트 하고자 하는 이벤트 명령어를 작성하는 공간이며, Fig. 5 이벤트 테이블에서 정의한 이벤트 명령어를 참조한다. 이벤트 명령어 실행은 각 모드의 수행 흐름에 맞춰 실행된다.

3.4 체크리스트 기반 테스트 조합 생성

앞에서 정의한 모드를 조합하여 다양한 테스트 스크립트를 생성할 수 있다. 하나의 동작 모드는 특정 동작을 위한 일련의 기능이다. 다양한 동작 순서에서 SUT가 정상적으로 동작하는지 테스트하기 위해서는 다양한 모드 조합으로 생성한 테스트 스크립트가 필요하다.

모드의 순서 조합에 따라 시스템이 도달하는 상태가 다르고 이에 따른 시스템의 동작 오류를 파악해야 할 필요가 있다. 따라서 본 논문에서는 모드의 조합을 각 모드의 고유 기능 테스트에 집중하는 순차적 방법, 기존의 Pairwise 및 Triple Pairwise 방법[20]에서 사용하는 방법과 같은 개념의 조합인 Double Permutation 과 Triple Permutation방법 그리고 무작위 조합에 따른 동작 확인을 위해 무작위 조합을 사용하는 방법 등 이전 연구 [29]에서와 같은 방법들을 사용할 수 있다.

순차적 방법은 사용자가 원하는 시나리오를 테스트하기 위해 모드의 순서를 선택하여 테스트 스크립트를 생성하는 방법이다. 순차지정파일에 작성된 순서대로 테스트 케이스를 생성한다.

Double Permutation 방법은 두 특정 모드가 차례로 수행되었을 때의 SUT 동작을 테스트하기 위한 테스트 스크립트를 생성하는 방법이다. 또한 각 동작 모드의 “Pre” 조건이나 “Post” 조건을 하나 이상을 갖는 경우 하나의 모드에서 다른 모드로 진입이 정상적으로 되는지, 다양한 “Pre” 조건 및 “Post” 조건에서 시스템이 정상적으로 동작하는지 테스트할 수 있다.

Triple Permutation은 Double Permutation 과 기본적인 생성 원리는 같다. 하지만 Double Permutation 과 같이 두 특정 모드가 차례대로 수행된 것이 아닌 세 특정 모드가 차례대로 수행되었을 때 SUT의 동작을 테스트 하기위한 테스트 스크립트를 생성한다.

무작위 생성은 Fig. 5에 작성된 이벤트 사전에서 이벤트를 무작위로 선택한다. 또한 무작위 생성 시 테스트 하고자 하는 이벤트 수와 각 이벤트 간 대기시간을 설정할 수 있다. 이벤트 사이에 대기시간을 정해주는 이유는 해당 이벤트가 수행되었을 때 SUT의 특성에 따라 명령어 유지 시간이 달라질 수 있기 때문이다.

3.5 테스트 스크립트 자동 생성 프로그램

앞 절에서 제안한 방법으로 구현한 테스트 스크립트 생성 프로그램의 초기 화면은 Fig. 9와 같다.

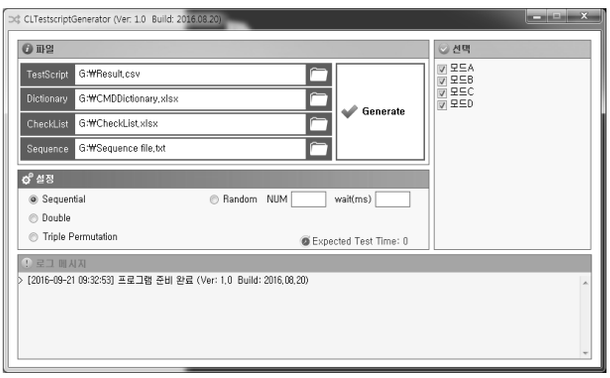


Fig. 9. Test Script Generator Screen Shot

Fig. 9 테스트 스크립트 자동생성 프로그램은 테스트 명령어 사전 파일, 체크리스트 파일, 순서 지정 파일을 입력받을 수 있다. 체크리스트 파일에 작성된 전체 모드를 화면 체크리스트 박스에 표시해주며, 테스트하고자 하는 모드를 테스트 엔지니어가 선택할 수 있다.

설정 옵션 중 Sequential 옵션은 순차지정 파일에 작성된 순서대로 테스트 스크립트를 생성한다. 만약 순차지정 파일을 입력하지 않은 경우 화면 체크리스트 박스에 선택된 순서대로 테스트 스크립트를 생성한다.

Double Permutation 옵션은 테스트 엔지니어가 선택한 모드를 조합하고, 각 모드마다 “Pre” 와 “Post” 조건에 조건 조합 정책을 적용하여 테스트 스크립트를 생성한다. Tripe Permutation은 앞서 설명한 Double Permutation과 같은 방식으로 테스트 스크립트를 생성하며, 세 개의 모드 조합을 테스트 한다.

Random 옵션은 입력된 커맨드 개수(NUM)과 대기시간(wait)에 따라 이벤트 명령어를 랜덤으로 선택하여 테스트 스크립트를 생성한다.

Expected Test Time은 생성된 테스트 스크립트를 실제로 수행하는데 걸리는 전체 시간을 표시해준다. 사용자는 표시된 시간을 보고 테스트 예상 시간을 판단할 수 있다. 마지막으로 로그 메시지는 테스트 스크립트 생성 결과 및 오류 정보를 사용자에게 표시해준다.

4. 실 험

본 논문에서 제안하는 방법을 기반으로 구현된 테스트 스크립트 자동 생성 프로그램이 테스트 스크립트를 올바르게 생성하는지 확인하기 위해 로그 파일을 이용하여 생성 과정 및 테스트 스크립트 결과를 분석하였다.

테스트 대상 시스템은 가상의 유도미사일이다. 실험에서 사용한 체크리스트 파일은 4개의 모드로 구분되며, 각 동작

모드의 이름은 유도모드, 일반모드, 특수모드, 대기모드로 정의하였다. Fig. 10은 4개의 모드 중 유도모드에 대한 체크리스트 구성을 나타낸 그림이다.

유도모드의 요구사항이 “유도 미사일이 고도 2km, 속도 400m/s 이상 도달 했을 때 유도 기능을 활성화 한다.” 라고 가정하면, Fig. 10과 같이 체크리스트 작성이 가능하다.

모드	기능설명	조건	이벤트
유도모드	미사일 엔진을 점화시킨다.	Pre	엔진점화
	미사일의 고도 센서가 정상적인 상태로 진입		고도정상
	미사일의 속도 센서가 정상적인 상태로 진입		속도정상
	미사일의 유도 기능을 시작한다.	Act	유도기능시작
	미사일의 유도 기능을 정상적으로 종료한다.	Post	유도기능종료
	미사일의 고도 센서 오류가 발생한 경우		고도센서오류
	미사일의 속도 센서 오류가 발생한 경우		속도센서오류

Fig. 10. Guided Mode Checklist

Fig. 10 유도모드 테이블의 “Pre” 조건은 3가지 조건으로 구성되어있다. “엔진점화”의 의미는 미사일의 엔진이 점화되었다는 의미이고, “고도정상”과 “속도정상”은 미사일 엔진이 점화된 후 고도와 속도가 정상적인 상태로 진입했다는 의미이다. “Act” 조건에 작성된 “유도기능시작”은 모든 “Pre” 조건이 만족할 경우 유도기능을 시작하겠다는 의미이다. “Post” 조건에 작성된 이벤트는 유도기능을 정상적으로 종료한 경우와 오류 발생으로 인해 비정상적으로 종료되는 경우의 이벤트를 작성한 것이다.

파일타입: 이벤트

이벤트	group	컴포넌트 참조
유도기능시작	1	모드(1)
		TIME(3600)
유도기능종료	1	모드(0)
엔진점화	2	발사신호감지장치(1)
		엔진장치(1)
엔진종료	2	엔진장치(0)
		발사신호감지장치(0)
고도정상	3	고도장치(5000)
고도비정상	3	고도장치(300)
고도센서오류	3	고도장치(-500)
속도정상	4	속도장치(500, 200)
속도비정상	4	속도장치(80, 50)
속도센서오류	4	속도장치(-50, -40)
전원켬	5	전원(1)
		TIME(1000)
전원끔	5	전원(0)
일반모드동작	6	모드(2)
일반모드종료	6	모드(0)
특수모드동작	7	모드(3)
특수모드종료	7	모드(0)
대기모드동작	8	모드(4)
대기모드종료	8	모드(0)

Fig. 11. The Event Dictionary of Guided Missile

Fig. 11 유도미사일 이벤트 사전은 Fig. 10 체크리스트에서 참조하는 모든 이벤트를 정의한 이벤트 테이블이다. 예를 들어 Fig. 10 체크리스트의 “이벤트” 열에는 “엔진점화”, “고도정상”, “속도정상” 등의 이벤트가 작성되어 있으며, 이러한 모든 이벤트는 Fig. 11에 정의된다. 예를 들면, Fig. 10에 정의된 다양한 이벤트 중 “엔진점화” 이벤트의 의미는 “엔진점화” 이벤트가 발생됨으로써 “발사신호감지장치”와 “엔진장치” 컴포넌트가 조작된다는 의미이다.

파일타입: 컴포넌트

컴포넌트	명령어 참조
엔진장치(#i)	속도(0)
	자이로(0)
	연료제어장치(1000, 1000)
	점화(#i)

컴포넌트	명령어 참조
연료제어장치(#fl, #fr)	연료(#fl, #fr)

컴포넌트	명령어 참조
발사신호감지장치(#f)	발사감지센서(#f)

컴포넌트	명령어 참조
고도장치(#alt)	기압(500)
	고도(#alt)

컴포넌트	명령어 참조
속도장치(#s, #g)	자이로(#g)
	TIME(100)
	속도(#s)

Fig. 12. The Component Dictionary of Guided Missile

Fig. 12 유도미사일 컴포넌트 테이블은 Fig. 11 유도미사일 이벤트에서 참조하는 모든 컴포넌트를 정의한 컴포넌트 사전이다. 각 컴포넌트는 자신이 사용하는 명령어를 포함한다.

파일타입: 명령어

명령어	TIME
TIME(#t)	#t

명령어	TIME	시스템 입력
		SysIn_Atm
기압(#a)	0	#a

명령어	TIME	시스템 입력
		SysIn_Power
전원(#p)	10	#p

명령어	TIME	시스템 입력
		SysIn_Gyro
자이로(#g)	30	10 * #g
	30	100 * #g
	30	1000 * #g

명령어	TIME	시스템 입력	
		SysIn_IgnitionA	SysIn_IgnitionB
점화(#g)	20	#g	10

명령어	TIME	시스템 입력	
		SysIn_ModeA	SysIn_ModeB
모드(#m)	20	#m	1

명령어	TIME	시스템 입력	
		SysIn_FuelSensorR	SysIn_FuelSensorL
연료(#l, #r)	10	-1	-1
	10	0	0
	10	#l	#r

명령어	TIME	시스템 입력
		SysIn_Plug
발사감지센서(#p)	10	#p

명령어	TIME	시스템 입력
		SysIn_Altitude
고도(#a)	5	-50
	5	1000 * #a

명령어	TIME	시스템 입력
		SysIn_Speed
속도(#s)	15	5000 + #s

Fig. 13. The Command Dictionary of Guided Missile

Fig. 13의 유도미사일 입력 명령어 사전은 Fig. 12 유도미사일 컴포넌트에서 참조하는 모든 명령어를 정의한 명령어 테이블이다. 이벤트부터 명령어 입력까지 호출되는 과정을 예는 다음 절에서 나온다.

Table 1. Number of Events for the Modes

	Pre	Act	Post
유도모드	3 (엔진점화, 고도정상, 속도정상)	1 (유도기능시작)	3 (유도기능종료, 고도센서오류, 속도센서오류)
일반모드	1 (전원켬)	1 (일반모드동작)	1 (일반모드종료)
대기모드	0	1 (대기모드동작)	1 (대기모드종료)
특수모드	0	1 (특수모드동작)	1 (특수모드종료)

Table 1은 실험에서 사용한 전체 모드(유도모드, 일반모드, 대기모드, 특수모드)의 “Pre”, “Act”, “Post” 조건에서 참조하는 모든 이벤트의 개수를 나타낸 테이블이며, 총 14개의 이벤트로 구성되어있다.

4.1 테스트 스크립트 생성

앞서 정의한 유도모드의 체크리스트와 이벤트, 컴포넌트, 입력 명령어 사전을 이용하여 유도모드의 테스트 스크립트 생성 과정은 다음과 같다.

체크리스트의 수행 흐름은 “Pre → Act → Post” 순서로 수행되며, Fig. 10 유도모드의 체크리스트 수행과정은 “Pre” 조건인 “엔진점화” 이벤트가 가장 먼저 수행된다. Fig. 11 이벤트 테이블의 “엔진점화” 이벤트 구성을 살펴보면, “발사신호감지장치”와 “엔진장치”로 이루어져있다. 이 의미는 “엔진점화” 이벤트를 발생시킴으로써 “발사신호감지장치”와 “엔진장치” 컴포넌트가 참조된다는 의미이다. “발사신호감지장치”의 인수는 “1” 값으로 정의되어있고, 이에 따라 Fig. 12 컴포넌트 테이블에 정의된 “발사신호감지장치”의 매개변수 “#f”는 “1”로 치환된다. “발사신호감지장치”의 명령어 참조에서는 “발사감지센서” 명령어를 참조하고 있다. “발사신호감지장치”의 매개변수 “1”은 다시 Fig. 13 명령어 사전 내의 “발사감지센서”의 매개변수 “#p” 로 전달된다. “발사감지센서”의 시스템입력으로 “SysIn_Plug”가 정의되어있고, 최종적으로 “SysIn_Plug” 시스템입력으로 인가되는 신호 “1”이 테스트 케이스로 생성될 것이다.

Fig. 14는 유도모드의 테스트 스크립트 생성결과이며, 총 14개의 테스트 케이스가 생성되었다. 테스트 케이스 “#1”은 초기 입력 값을 정의한 “InitValue”가 수행되어 생성된 테스트 케이스이다. 다음 테스트 케이스인 “#2”는 앞서 설명한 “엔진점화” 이벤트에 정의된 “발사신호감지장치(1)”이 수행되어 생성된 테스트 케이스이며, “SysIn_Plug” 값이 “1”로 변화된 것을 확인할 수 있다.

테스트 케이스 “#3”은 “엔진점화” 이벤트에 정의된 “엔진장치(1)”이 수행되었고, 이에 따라 엔진장치에 정의된 “속도(0)” 명령어가 순차적으로 수행되어 생성된 테스트 케이스이다. 명령어 테이블의 “속도”는 “SysIn_Speed” 시스템 입력을 정의하고 있으며, “SysIn_Speed”의 입력 신호는 “5000 + #s”으로 정의되어있다. “#s” 값은 앞서 정의한 “속도(0)”의 매개변수 “0” 값으로 치환되었고, “5000 + 0”의 계산결과에 따라 테스트 케이스 “#3”의 “SysIn_Speed” 시스템 입력 값이 “5000”으로 생성된 것을 확인할 수 있다.

모든 이벤트 처리는 앞서 설명한 것과 같은 방식으로 처리된다. 테스트 케이스 “#2” 부터 “#6” 까지는 “엔진점화” 이벤트가 수행되어 생성된 테스트 스크립트이고, “#7”부터 “#8” 까지는 “고도정상” 이벤트가 수행되어 생성된 테스트 스크립트이다. 마찬가지로 “#9”부터 “#11” 까지는 “속도정상” 이벤트가 수행되어 생성된 것이고, “#12”부터 “#13” 까지는 “유도기능시작” 이벤트가 수행되어 생성된 것이다. 마지막으로 “#14”는 “유도기능종료” 이벤트가 수행되어 생성된 테스트 스크립트이다.

“Pre → Act → Post”의 순서에 따라 “엔진점화 → 고도정상 → 속도정상 → 유도기능시작 → 유도기능종료”의 순서로 테스트 스크립트가 생성되었다. “Post” 조건은 각 모드가 호출될 때마다 순차적으로 하나의 이벤트가 수행된다. 만약 이후에 유도모드가 다시 호출된다면, 두 번째 작성된 “고도센서오류” 이벤트가 수행될 것이다.

각 테스트 케이스의 “TIME”은 Fig. 13 명령어 테이블에 정의된 “TIME”을 참조하여 생성된 시간 값들이다. 또한 테스트 스크립트 결과에 작성된 “-” 표시는 이전 값을 유지하라는 의미이며, 이전 값으로부터 갱신되지 않은 데이터들이다.

각 이벤트의 처리 과정과 생성 결과를 비교해 보았을 때, 테스트 케이스가 모두 정상적으로 생성된 것을 확인할 수 있었다.

	SysIn_Power	SysIn_IgnitionA	SysIn_IgnitionB	SysIn_FuelSensorR	SysIn_FuelSensorL	SysIn_Altitude	SysIn_Speed	SysIn_Atm	SysIn_Gyro	SysIn_ModeA	SySIn_ModeB	SysIn_Plug	TIME
#1	1	0	0	0	0	0	0	0	0	0	0	0	0
	-	-	-	-	-	-	-	-	-	-	-	-	10000
#2	-	-	-	-	-	-	-	-	-	-	-	1	10000
#3	-	-	-	-	-	-	5000	-	-	-	-	-	15000
#4	-	-	-	-	-	-	-	-	0	-	-	-	30000
	-	-	-	-	-	-	-	-	0	-	-	-	30000
	-	-	-	-	-	-	-	-	0	-	-	-	30000
#5	-	-	-	-1	-1	-	-	-	-	-	-	-	10000
	-	-	-	0	0	-	-	-	-	-	-	-	10000
	-	-	-	1000	1000	-	-	-	-	-	-	-	10000
#6	-	1	10	-	-	-	-	-	-	-	-	-	20000
#7	-	-	-	-	-	-	-	500	-	-	-	-	0
#8	-	-	-	-	-	-50	-	-	-	-	-	-	5000
	-	-	-	-	-	5000000	-	-	-	-	-	-	5000
#9	-	-	-	-	-	-	-	-	2000	-	-	-	30000
	-	-	-	-	-	-	-	-	20000	-	-	-	30000
	-	-	-	-	-	-	-	-	200000	-	-	-	30000
#10	-	-	-	-	-	-	-	-	-	-	-	-	100000
#11	-	-	-	-	-	-	5500	-	-	-	-	-	15000
#12	-	-	-	-	-	-	-	-	-	1	1	-	20000
#13	-	-	-	-	-	-	-	-	-	-	-	-	3600000
#14	-	-	-	-	-	-	-	-	-	0	1	-	20000

Fig. 14. The Result of Test Script Generation for the Guided Mode

4.2 순차적 모드 생성

본 실험에서는 Table 1에 작성된 전체 모드를 순차적 모드(Sequential)로 생성할 때, 순차지정 파일에 작성된 테스트 스크립트의 통계를 확인한다.

Fig. 15는 순차적 모드 생성에서 사용한 순차지정 파일이다. 모드 수행 순서는 초기 “유도모드” 다음은 항상 “일반모드” 수행된다. 여기서 “유도모드”의 “Post” 조건은 항상 “속도센서오류”를 수행한다. 마찬가지로 “특수모드” 다음은 항상 “유도모드” 수행되고, “대기모드” 다음은 항상 “일반모드”가 수행된다.

```
"유도모드_속도센서오류/일반모드";
"특수모드/유도모드";
"대기모드/일반모드";
```

Fig. 15. Sequence File

테스트 생성 프로그램은 테스트 스크립트 생성 시 이벤트 발생 순서를 나열한 디버그 파일을 함께 생성한다.

1	엔진점화	#1 유도모드	11	엔진점화	#4 유도모드
2	고도정상		12	고도정상	
3	속도정상		13	속도정상	
4	유도기능시작		14	유도기능시작	
5	속도센서오류		15	유도기능종료	
6	전원켬	#2 일반모드	16	대기모드동작	#5 대기모드
7	일반모드동작		17	대기모드종료	
8	일반모드종료		18	전원켬	
9	특수모드동작	#3 특수모드	19	일반모드동작	#6 일반모드
10	특수모드종료		20	일반모드종료	

Fig. 16. The Event Sequence of Sequential Mode

Fig. 16는 테스트 스크립트 생성 시 함께 생성되는 디버그 파일이다. Fig. 15 순차지정 파일에 작성한 내용과 같이 테스트 스크립트 생성 순서는 “#1 유도모드 → #2 일반모드 → #3 특수모드 → #4 유도모드 → #5 대기모드 → #6 일반모드” 순서로 생성되었다. 또한 “#1 유도모드”의 5번째 행에 “속도센서오류” 이벤트가 생성되었다. 이 이벤트는 순차지정 파일에 작성한 “유도모드_속도센서오류”가 수행되어 생성된 “Post” 이벤트이다.

각 모드의 “Pre”, “Act”, “Post” 조건을 고려한 이벤트 처리 및 테스트 스크립트 생성 과정은 4.1절에서 설명한 내용과 동일하다.

이벤트가 참조한 컴포넌트와 입력 명령어는 Fig. 12와 Fig. 13이다. 사용된 컴포넌트 수는 5개, 컴포넌트가 참조한 입력 명령어 내의 입력 신호 인가 수는 10개이다. 제안한 구조에서는 Fig. 16와 같이 시스템 정보가 전혀 없는 추상화된 간단한 20개의 이벤트를 사용하여 Fig. 15의 기능을 테스트할 수 있으나, 제안한 방법이 없었다면 Fig. 17과 같이 모든 입력 정보로 구성된 24개의 복잡한 명령어를 테스트 엔지니어가 작성해야 한다는 의미이다. 물론 처음 컴포넌트와 입력 명령어 사전 작성 시에 소요되는 노력은 같지만, 이후 이들 사전을 사용하여 테스트 스크립트를 생성할 경우는 시간과 노력을 획기적으로 단축될 수 있다.

이벤트	참조 컴포넌트 및 명령어		
	상위 컴포넌트	하위 컴포넌트	명령어
엔진점화	발사신호감지장치	-	발사감지센서
	엔진장치	-	속도
		-	자이로
		연료제어장치	연료
		-	점화
고도정상	고도장치	-	기압
		-	고도
속도정상	속도장치	-	자이로
		-	TIME
		-	속도
유도기능시작	-	-	모드
	-	-	TIME
유도기능종료	-	-	모드
속도센서오류	속도장치	-	자이로
		-	TIME
		-	속도
전원켬	-	-	전원
	-	-	TIME
일반모드동작	-	-	모드
일반모드종료	-	-	모드
특수모드동작	-	-	모드
특수모드종료	-	-	모드
대기모드동작	-	-	모드
대기모드종료	-	-	모드

Fig. 17. The Commands of the Sequential Mode Event

이벤트	group	컴포넌트 참조
엔진점화	2	발사신호감지장치(1)
		엔진장치(1)

컴포넌트 추가

이벤트	group	컴포넌트 참조
엔진점화	2	발사신호감지장치(1)
		속도장치(0, 0)
		고도장치(0, 0)
		엔진장치(1)

Fig. 18. An Example of Components Addition in “엔진점화” Event

또한 Fig. 18과 같이 새로운 컴포넌트나 입력 명령어를 추가할 경우 새로운 기능 테스트에 활용할 수 있어 한번 작성된 사전은 재활용성과 확장성이 매우 높다. 예를 들면, Fig. 19와 같이 테스트 순서 및 기능을 바꿀 경우에는 새로운 테스트 명령어 사전들을 작성할 필요 없이 이미 만들어진 사전들을 참조하면 되고, Fig. 20과 같은 이벤트만을 생성하면 된다. 이 때, 엔지니어는 컴포넌트 계층 이하 하드웨어에 가까운 기능을 정확히 이해하지 않아도 테스트 스크립트 생성이 가능하게 된다.

```
"유도모드/특수모드";
"일반모드/대기모드/특수모드";
```

Fig. 19. Modified Sequence File

1	엔진점화	#1 유도모드	8	전원켬	#3 일반모드
2	고도정상		9	일반모드동작	
3	속도정상		10	일반모드종료	
4	유도기능시작		11	대기모드동작	#4 대기모드
5	유도기능종료		12	대기모드종료	
6	특수모드동작	#2 특수모드	13	특수모드동작	#5 특수모드
7	특수모드종료		14	특수모드종료	

Fig. 20. The Event Sequence for Fig. 19

4.3 Double Permutation 모드 생성

본 실험에서는 Table 1에 작성된 전체 모드를 MCDC 커버리지 기준[21]으로 Double Permutation을 생성할 때, 테스트 스크립트가 올바른 순서로 생성되는지 확인한다. 또한 실행 불가능한 Double Permutation 모드의 조합은 Fig. 21과 같이 파일 설정을 이용하여 조합을 배제할 수 있다. Fig. 21 파일에 작성된 "!특수모드/대기모드"의 의미는 특수모드 이후에 대기모드가 실행될 수 없다는 의미이다.

```
"!특수모드/대기모드";
"!대기모드/유도모드";
```

Fig. 21. Exclusive Mode

Fig. 22는 Double Permutation을 적용하였을 때, 유도모드의 이벤트 처리 순서를 나타낸 디버그 파일이다. <A> 구간은 "Pre" 조건에 MCDC 기법을 적용한 구간이며, <가>, <나>, <다>에서는 해당 이벤트가 TRUE 로 동작하는 경우와 FALSE 로 동작하는 경우를 표시하였다. 이에 따라 "Pre" 조건인 "엔진점화", "고도정상", "속도정상" 이벤트는 각 참과 거짓에 맞는 테스트 케이스를 생성한다. 이러한 테스트 케이스는 모든 "Pre" 조건이 정상적인 경우에 "Act"에 정상 진입할 수 있을 것이다.

<B> 구간은 모든 "Pre" 조건과 "Act" 조건이 참인 경우에 모든 "Post" 조건을 테스트 하는 구간으로 유도모드의 3개 "Post" 조건(유도기능종료, 고도센서오류, 속도센서오류)이 모두 수행되도록 테스트 케이스를 생성한다. 앞서 설명한 기법은 "유도모드" 뿐만 아니라 "일반모드", "특수모드", "대기모드"에 모두 적용된다.

Fig. 22는 28개의 추상화된 이벤트만으로 Double Permutation의 유도모드 테스트 스크립트를 작성할 수 있음을 보여준다. 이들 이벤트가 참조한 컴포넌트는 5개, 그리고 사용된 입력 명령어 개수는 총 10개이다. 이전에 작성된 테스트 사전을 활용하면 입력 명령어 작성에 직접 개입하는 기존 방법보다 테스트 스크립트 생성에 드는 노력이 획기적으로 줄어듦을 알 수 있다.

<A>	1	InitValue		
	2	=!엔진점화		FALSE
	3	고도정상		TRUE
	4	속도정상		TRUE
	5	유도기능시작		TRUE
	6	엔진점화		TRUE
	7	=!고도정상		FALSE
	8	속도정상		TRUE
	9	유도기능시작		TRUE
	10	엔진점화		TRUE
	11	고도정상		TRUE
	12	=!속도정상		FALSE
	13	유도기능시작		TRUE
<B>	14	엔진점화		TRUE
	15	고도정상		TRUE
	16	속도정상		TRUE
	17	유도기능시작		TRUE
	18	유도기능종료		
	19	엔진점화		TRUE
	20	고도정상		TRUE
	21	속도정상		TRUE
	22	유도기능시작		TRUE
	23	고도센서오류		
	24	엔진점화		TRUE
	25	고도정상		TRUE
	26	속도정상		TRUE
	27	유도기능시작		TRUE
	28	속도센서오류		

<가>: 1~13, <나>: 6~13, <다>: 10~13

Fig. 22. Event Sequence of the Double Permutation

5. 결 론

본 논문에서는 체크리스트를 체계적으로 구조화하고, 구조화된 데이터로부터 테스트 스크립트를 효율적으로 생성하는 방법을 제안하였다.

테스트 스크립트를 자동으로 생성하기 위해 제안하는 방법은 계층적 구조를 가지며, 각 계층은 이벤트, 컴포넌트, 명령어 및 시스템 입력으로 구성된다.

가상 유도미사일의 테스트 스크립트 생성 실험을 통하여 제안하는 구조의 장점을 확인하였다. 3종류의 테스트 사전을 활용하여 테스트 스크립트를 생성할 경우 추상화된 이벤트를 활용하여 시스템 입력에 대한 자세한 지식 없이도 테스트 스크립트 생성이 가능함을 보였다. 또한 각 계층의 사전은 독립적으로 쉽게 수정이 가능하여 사전 관리가 매우 쉬우며 재활용성이 매우 높다. 체크리스트로부터 체계적인 테스트 스크립트를 생성하는 방법이 이전 연구에 많지 않아 기법 비교가 어렵지만, 여러 실험을 통해 테스트 엔지니어가 예측하기 어려운 다양한 테스트 스크립트 생성이 가능하다는 것을 확인하였다.

좀 더 크고 복잡한 시스템 테스트 스크립트 사용을 통한 확인 등의 추후 과제가 남아 있지만 제안하는 구조의 편리성과 효용성은 충분히 있다고 판단된다.

References

[1] J. Y. Seo, A. Y. Sung, B. J. Choi, and S. B. Kang, "Automating Embedded software Testing on an Emulated Target Board," *Proc. of the Second International Workshop on Automation of Software Test*, 20-26 May, 2007.

[2] S. Y. Jeong, Y. W. Chang, and C. J. Yoo, "Test Case Genration Technique Based on State Transition Model for Embedded System," *Journal of Korean Institute of Information Technology*, Vol.9, No.4, pp.11-21, 2011.

[3] M. R. Keyvanpour, H. Homayouni, and Hossein Shirazee, "Automatic Software Test Case Generation: An Analytical Classification Framework," *International Journal of Software Engineering and Its Applications*, Vol.6, No.4, pp.1-16, October, 2012.

[4] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedigns of the 9th Asian Computaing Science Conference*, volum 3321 of Lecture Notes in Computer Science, pp.320-329, 2004.

[5] Kuo Chung Tai and Yu Lei, "A Test Generation Strategy for Pairwise Testing," *IEEE Transactions on Software Engineering*, Vol.28, No.1, pp.109-111, January, 2002.

[6] K. P. Chan, T. Y. Chen, and Dave Towey, "Restricted Random Testing," in *Proceedings of the 7th European Conference on Software Quality Helsinki,* Finland, Vol.2349/2002 of Lecture Notes in Computer Science, pp.321-330, June, 2002.

[7] Conrad, M., Dörr, H., Fey, I., and Yap, A., "Model-based Generation and Structured Representation of Test Scenarios," Workshop on Software-Embedded Systems Testing (WSEST), Gaithersburg, USA, November, 1999.

[8] P. S. Loo and W. K. Tsai, "Random Testing Revisited," Information and Software Technology, Vol.30, Issue 7, pp.402-417, Sep., 1988.

[9] T. Y. Chen, F. C. Kuo, Huai Liu, and W. E. Wong, "Does Adaptive Random Testing Deliver a Higher Confidence than Random Testing?," *The Eighth International Conference on Quality Software*, 2008, QSCI'08, pp.145-154, 12-13 August, 2008.

[10] M. Grochtmann and K. Grimm, "Classification Trees for Partition Testing," *Software Testing, Verification & Reliability*, Vol.3, No.2, pp.63-82, June, 1993.

[11] J. H. Shin, K. H. Chung, and K. H. Choi, "Destructive Test of a BLDC Motor controller utilizing a Modified Classification Tree Method," *KIPS Tr. Software and Data Eng.,* Vol.3, No.6 pp.201-214, piSSN: 2287-5905, March 21, 2014.

[12] P. M. Kruse and M. Luniak, "Automated test case generation using classification trees," *Software Quality Professional Magazine*, Vol.13, No.1, pp.4-12, 2010.

[13] The International Engineering Consortium, technical report, "Specification and Description Language(SDL)."

[14] AGEDIS Consortium, technical report, "Model Based Test Generation Tools."

[15] A. Hartman and K. Nagin, "The AGEDIS Tools for Model Based Testing," *Proceedings of the 2004 ACM SIGSOFT International symposium on software testing and analysis*, pp.129-132.

[16] Alexander Pretschner, "Model-Based Testing," *Proceedings of the 27th interantional conference on Software engineering*, pp.723-822.

[17] H. S. Park, "Generating Structural Test Cases for MATLAB Stateflow Model Using Rapidly-exploring Random Tree," Ajou Univ, Engineering doctoral dissertation, 2014.

[18] M. Utting and B. Legeard, "Practical Model-Based Testing: A Tools Approach," Morgan kaufmann, 2007.

[19] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson, "A Practical Tutorial on Modified Condition/Decision Coverage," NASA, 2001.

[20] Yu Lei and K. C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," in *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium*, pp.254-261, 1998.

[21] A. P. Mathur, "Foundations of Software Testing," Pearson Education, 2008.

[22] Junyeon Hwang, "Auto Test Script Generation Based on Checklist," Master Dissertation, Ajou University, Suwon, Korea, 2015.

[23] Joseph Schmuller, "Teach yourself UML in 24 Hours, 3/E," SAMS, 2004.

[24] Ivar Jacobson, "Object-Oriented Software Engineering: A Use-Case-Driven Approach," Addison-Wesley, 1992.

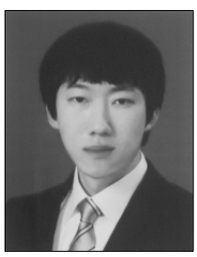
[25] Lvar Jacobson, Kurt Bittner, lan Spence, "Use Case Modeling," Addison-Wesley, 2002.

[26] Paul C. Jorgensen, "Software Testing: A Craftsman's Approach, 4/E," CRC Press, 2016

[27] U.S.NRC, "Fault Tree Handbook (NUREG-0492)," US, 1981

[28] Rick Kuhn, Raghu Kacker, Yu Lei, and Justin Hunter, "Combinatorial Software Testing," *IEEE Computer Society*, Vol.42, Issue 8, pp.94-96, August 2009.

[29] Kang Tae Hoon, Kim Dae Joon, Chung Ki Hyun, and Choi Kyung Hee, "A Method to Automatically Generate Test Scripts from Checklist for Testing Embedded System," KIPS, 2016 to be published.



김 대 준

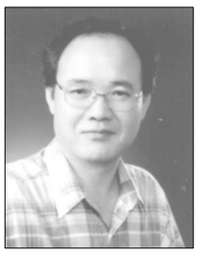
e-mail : kdj4527@ajou.ac.kr

2015년~현 재 아주대학교 컴퓨터공학과 석사과정

관심분야 : 운영체제, 임베디드 시스템, 임베디드 시스템 테스팅 등

정 기 현

e-mail : khchung@ajou.ac.kr
1984년 서강대학교 전자공학과(학사)
1988년 미국 Ilinois주립대 EECS(석사)
1990년 미국 Purdue대학 전기전자공학부 (박사)
1991년～1992년 현대반도체 연구소
1993년～현 재 아주대학교 전자공학과 교수
관심분야 : 컴퓨터구조, VLSI 설계, 멀티미디어/실시간 시스템



최 경 희

e-mail : khchoi@ajou.ac.kr
1976년 서울대학교 수학교육과(학사)
1979년 프랑스 그랑데꼴 Enseeiht 대학 (석사)
1982년 프랑스 Paul Sabatier 대학 정보공학부(박사)
1982년～현 재 아주대학교 컴퓨터공학과 교수
관심분야 : 운영체제, 분산시스템, 실시간/멀티미디어 시스템