Journal of the KIMST, Vol. 23, No. 4, pp. 389-398, 2020
ISSN 1598-9127(Print) · 2636-0640(Online)
DOI https://doi.org/10.9766/KIMST.2020.23.4.389

Research Paper 정보·통신 부문

무기체계 소프트웨어의 모델 기반 테스트 케이스 생성 방법

최현재[1] · 이영우[1] · 백지선[2] · 김동환[3] · 조규태[2] · 채흥석[*,1]

[1] 부산대학교 컴퓨터공학과
[2] LIG넥스원㈜ SW지능화연구팀
[3] LIG넥스원㈜ 지능형SW연구소

Model-based Test Cases Generation Method for Weapons System Software

Hyunjae Choi[1] · Youngwoo Lee[1] · Jisun Baek[2] · Donghwan Kim[3] · Kyutae Cho[2] · Heungseok Chae[*,1]

[1] *Department of Computer Engineering, Pusan National University, Korea*
[2] *Intelligent Software System, LIG Nex1, Korea*
[3] *Intelligent & Software, LIG Nex1, Korea*

(Received 8 May 2020 / Revised 7 July 2020 / Accepted 24 July 2020)

Abstract

Test cases in the existing weapon system software were created manually by the tester analyzing the test items defined in the software integration test procedure. However, existing test case generation method has two limitations. First, the quality of test cases can vary depending on the tester's ability to analyze the test items. Second, excessive time and cost may be incurred in writing test cases. This paper proposes a method to automatically generate test cases based on the requirements model and specifications to overcome the limitations of the existing weapon system software test case generation. Generate test sequences and test data based on the use case event model, a model representing the requirements of the weapon system software, and the use case specification specifying the requirements. The proposed method was applied to 8 target models constituting the avionics control system, producing 30 test sequences and 8 test data.

Key Words : Model based Testing(모델 기반 테스팅), Test Case Generation(테스트 케이스 생성), Test Automation(테스트 자동화), Waepon System Software Test(무기체계 소프트웨어 테스트)

1. 서 론

소프트웨어 개발 산업에서 소프트웨어 테스팅은 전체 소프트웨어 개발 비용의 약 50 %를 차지하는 많은 비용이 요구되는 작업이다[1]. 그리고 소프트웨어 테스팅은 테스트 케이스 생성, 테스트 실행, 테스트 결과

* Corresponding author, E-mail: hschae@pusan.ac.kr

Table 1. Reference model by analysis phase

분석 단계	세부 모델명	설명
체계 요구사항 분석	System Interface Model	시스템과 액터, 데이터 스토어 사이의 인터페이스를 정의하고, 입/출력 데이터에 대한 이름, 입/출력 유형, 설명을 정의
	System Capability Model	시스템이 제공하는 기능 요구사항을 유스케이스를 이용하여 정의
소프트웨어 요구사항 분석	CSCI Interface Model	CSCI의 Port를 통해 제공하거나 요구되는 인터페이스를 정의하고, 입/출력 데이터의 이름, 입/출력 유형, 설명을 정의
	CSCI Capability Model	CSCI가 제공하는 소프트웨어 수준의 기능 요구사항을 유스케이스를 이용하여 정의
데이터 모델	System Data Conceptual Model	시스템의 동작을 통해서 생성되고 관리되는 데이터 스토어를 정의
	CSCI Data Conceptual Model	CSCI의 동작을 통해서 생성되고 관리되는 데이터 스토어를 정의

검증의 3단계로 이루어지는데, 그 중 테스트 케이스 생성은 전체 소프트웨어 테스팅 비용의 약 40 %를 차지한다[2]. 따라서 테스트 케이스 생성 비용 절감은 전체 소프트웨어 개발 비용 절감을 위해 중요한 이슈이다.

기존 무기체계 소프트웨어에서의 테스트 케이스는 테스터가 소프트웨어통합시험절차서[3]에 정의된 시험 항목을 분석하여 수동으로 생성하였다. 시험 항목은 관련 요구사항, 선행조건, 입력, 예상 결과, 평가 기준, 시험 절차, 가정 및 제약사항으로 구성된다.

그러나 기존 테스트 케이스 생성 방법은 다음 두 가지 한계점을 가진다. 첫째, 시험 항목을 분석하는 테스터의 역량에 따라 테스트 케이스의 품질이 달라질 수 있다. 둘째, 테스트 케이스 작성에 과도한 시간 및 비용이 발생할 수 있다.

본 연구는 기존 무기체계 소프트웨어 테스트 케이스 생성의 한계점을 극복하기 위해 무기체계 소프트웨어의 요구사항/설계 모델인 무기체계 소프트웨어 참조 모델을 기반으로 테스트 케이스를 자동 생성하는 방법을 제안한다. 무기체계 소프트웨어 참조 모델을 구성하는 요구사항 표현 모델인 유스케이스 이벤트 모델과 요구사항을 명세한 유스케이스 명세를 기반으로 테스트 시퀀스와 테스트 데이터를 생성한다. 유스케이스 명세의 메인/대안/예외 시나리오를 분석하여 테스트 시퀀스를 생성하고 유스케이스 이벤트 모델과 유스케이스 명세를 기반으로 테스트 데이터를 생성한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 배경 지식인 유스케이스 이벤트 모델과 무기체계 참조 모델을 소개한다. 3장에서는 유스케이스 이벤트 모델 기반 테스트 케이스 생성 방법을 제안한다. 4장에서는 사례 적용을 설명하고 5장에서는 관련 연구를 소개한다. 6장에서는 결론을 설명한다.

2. 배경 지식

본 장에서는 테스트 케이스 생성에 사용되는 유스케이스 이벤트 모델과 무기체계 소프트웨어 참조 모델을 소개한다. 유스케이스 이벤트 모델은 논리적인 시스템의 기능단위인 유스케이스를 정형화된 형태로 정의한 모델이며, 무기체계 소프트웨어 참조 모델은 방위사업청 무기체계 소프트웨어 모델을 UML 기반으로 정의한 참조 모델이다.

2.1 유스케이스 이벤트 모델

유스케이스 이벤트 모델은 UML의 유스케이스 모델을 기반으로 요구사항의 중요한 요소 중의 하나인 데이터 스토어, 그리고 유스케이스와 액터, 데이터 스토어 사이의 입/출력 데이터를 표현할 수 있는 모델이다[4,5]. 데이터 스토어는 시스템의 동작을 통해서 생성되고 관리되는 Persistent 데이터이다. Fig. 1은 유스케이스 이벤트 모델의 예이다.

유스케이스와 액터, 시스템 데이터 사이의 관계는 information flow로 표현하며, 각 flow에서 사용되는 flow item은 입/출력 유형에 따라 스테레오타입을 부여한다. 입력은 <<in>>, 출력은 <<out>>, 입출력은 <<inout>> 스테레오타입을 부여한다.

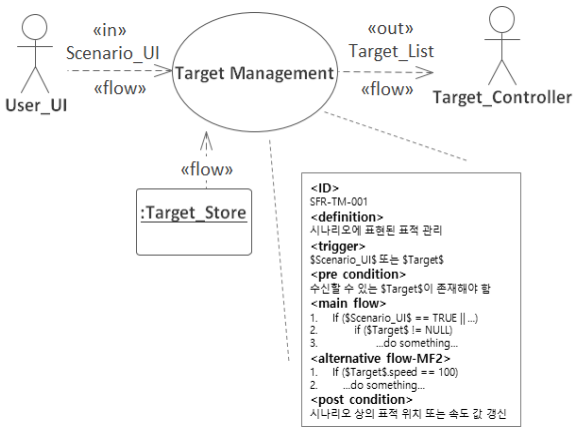


Fig. 1. Use case event model example

Fig. 1의 Scenario_UI는 User_UI에 의한 유스케이스의 입력이며, Target_List는 Target_Controller에 대한 유스케이스의 출력이다. 그리고 Target_Store 데이터 스토어로부터 유스케이스로 데이터가 입력된다.

유스케이스 명세는 텍스트 형태로 기술하며, <ID>, <definition>, <trigger>, <pre condition>, <post condition>, <main flow>, <alternative flow>, <exception flow>를 기술한다. 유스케이스 명세 중 <main flow>, <alternative flow>, <exception flow>는 각각 메인/대안/예외 시나리오를 정의한 것이다. 시나리오 명세에 사용되는 키워드는 Table 2와 같다.

Table 2. Scenario specification keyword

Type	Keyword
Conditional Statement	IF/ELIF/ELSE/ENDIF, LOOP/.ENDLOOP, FOR/ENDFOR, SWITCH/CASE/ENDSWITCH
Jump Condition	BREAK, CONTINUE, RETURN
Reference Variable	$variable$

<alternative flow>, <exception flow>는 “MF+숫자” 형태로 분기된 <main flow>의 스텝을 기술한다. 예를 들어, <main flow>의 2번째 스텝에서 <alternative flow>가 분기되는 경우, <alternative flow-MF2>와 같이 기술한다.

2.2 무기체계 소프트웨어 참조 모델

무기체계 소프트웨어 참조 모델[6]은 방위사업청 무기체계 소프트웨어 개발 프로세스에 명시된 개발 절차 중 체계요구사항분석, 체계구조설계, 소프트웨어요구사항분석, 소프트웨어구조/상세설계 단계에서 작성해야 하는 무기체계 소프트웨어 모델을 UML 기반으로 정의한 것이다. 무기체계 참조 모델은 체계요구사항명세서, 체계설계기술서, 소프트웨어요구사항명세서, 소프트웨어설계기술서 일부 구성 항목을 표현한다.

테스트 케이스 생성의 입력으로 사용되는 유스케이스 이벤트 모델은 체계요구사항분석 단계와 소프트웨어요구사항분석 단계에서 정의되며, 체계요구사항과 소프트웨어요구사항에서 사용되는 데이터는 데이터 모델에서 정의된다. 각 분석 단계 별 참조 모델의 구조는 Table 1과 같다.

유스케이스 이벤트 모델은 모델링 대상(System/CSCI)에 따라 System/CSCI Capability Model(Use Case Event Model)로 정의된다. Fig. 1의 Scenario_UI, Target, Target_List와 같은 시스템 입/출력 데이터(flow item)는 System/CSCI Interface Model에 정의된다. 예를 들어, System Interface Model은 Fig. 2와 같이 인터페이스를 통해 입/출력되는 데이터(flow item)를 클래스로 정의한다.

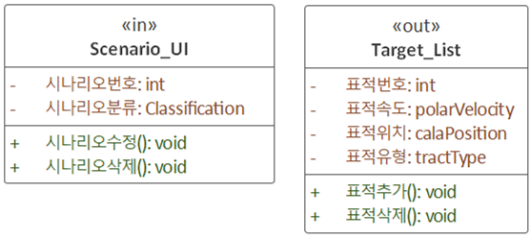


Fig. 2. System interface model example

System/CSCI에서 생성되고 관리되는 데이터 스토어는 System/CSCI Data Conceptual Model에 정의된다. Fig. 3은 시스템 데이터 스토어를 정의한 System Data Conceptual Model의 예를 보여준다.

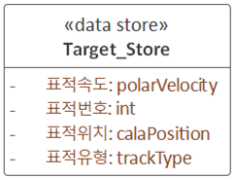


Fig. 3. System data conceptual model example

데이터 스토어는 <<data store>> 스테레오타입을 부여한 클래스로 정의된다. 각 데이터 스토어의 구체적인 속성은 클래스의 attribute로 표현한다.

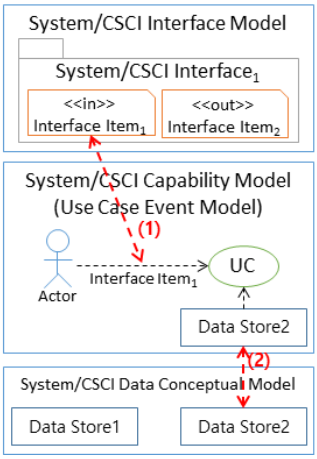


Fig. 4. System/CSCI requirement analysis reference model

각 참조 모델은 서로 다른 목적으로 작성되지만, 하나의 시스템을 대상으로 작성되므로, 참조 모델 간에는 만족해야 할 일관성이 존재한다. 예를 들어, Fig. 4의 참조 모델에서 (1) System./CSCI Capability Model에서 사용한 flow item은 System/CSCI Interface Model의 interface item과 일관성이 존재해야 하며, (2) System/CSCI Capability Model의 데이터 스토어는 System/CSCI Data Conceptual Model의 데이터 스토어와 일관성이 존재해야 한다.

3. 유스케이스 이벤트 모델 기반 테스트 케이스 생성 방법

본 장에서는 유스케이스 이벤트 모델과 유스케이스 명세를 기반으로 테스트 케이스를 생성하는 방법을 정의한다. 테스트 케이스 생성 방법은 테스트 시퀀스 생성과 테스트 데이터 분석으로 구성된다.

3.1 테스트 시퀀스 생성

테스트 시퀀스 생성은 테스트 모델 구축과 테스트 시퀀스 추출로 구성된다. 테스트 모델 구축은 유스케이스 명세의 기본/대안/예외 시나리오를 분석하여 테스트 모델을 자동 생성하는 단계이다. 테스트 시퀀스 추출은 테스트 모델의 경로를 추출하여 테스트 시퀀스를 생성하는 단계이다.

Table 3. Use case scenario grammar

```
TranslationUnit ::= Statement+

Statement ::= ( ConditionalStatement |
JumpStatement | NonConditionalStatement )
<NEW_LINE>

ConditionalStatement ::= ( IfStatement |
SwitchStatement | ForStatement | LoopStatement )

IfStatement ::= <IF> <STRING>
  <NEW_LINE> ( Statement )*
  [ ElseIfStatement ]* <ENDIF>

ElseIfStatement ::= <ELIF> <STRING>
  <NEW_LINE> ( Statement )*

SwitchStatement ::= <SWITCH> <STRING>
  <NEW_LINE> ( Statement )*
  [ CaseStatement ]+ <END_SWITCH>

CaseStatement ::= <CASE> <STRING>
  <NEW_LINE> ( Statement )*

ForStatement ::= <FOR> <STRING>
  “;” <STRING> “;” <STRING>
  <NEW_LINE> ( Statement )* <END_FOR>

LoopStatement ::= <LOOP> <STRING>
  “;” <STRING> “;” <STRING>
  <NEW_LINE> ( Statement )* <END_LOOP>

JumpStatement  ::= ( BreakStatement |
ContinueStatement | ReturnStatement )

BreakStatement ::= <BREAK> <STRING>

ContinueStatement ::= <CONTINUE> <STRING>

ReturnStatement ::= <RETURN> <STRING>

NonConditionalStatement ::= <STRING>
```

3.1.1 테스트 모델 구축

테스트 모델 구축은 시나리오의 제어 흐름을 분석하여 그래프 형태의 모델을 생성하는 단계이다. Table 2의 키워드를 이용하여 작성된 시나리오의 조건문과 점프문을 분석하여 액티비티 다어이그램을 생성한다.

테스트 시나리오 분석을 위해 시나리오 문법을 정의한다. 실세계의 소프트웨어 개발에서는 요구사항 분석 단계에서 정형화된 수준의 구체적인 시나리오 명세가 어렵기 때문에 시나리오 문법은 요구사항 정의의 조건문, 점프문 이외에는 제약을 두지 않고 자연어로 기술할 수 있도록 한다. Table 3은 정의한 유스케이스 시나리오 문법이다.

<STRING>은 키워드를 제외한 문자열이다. 자연어 표기를 허용하고 “;”와 같은 문장의 마침표를 사용하지 않는 대신, 조건식과 문장 구분을 위해 개행 문자인 <NEW_LINE>을 이용한다.

시나리오 문법을 기반으로 시나리오를 파싱하여 테스트 모델을 구축한다. Table 4는 예를 들어 설명할 테스트 시나리오 예제이다.

Table 4. Test scenario example

Scenario Name	Specification
main flow	1. IF (Cond1) 2. Stmt1 3. ENDIF 4. Stmt2
alternative flow-MF0	1. IF (CondAlt1) 2. StmtAlt1 3. ENDIF
exception flow-MF1	1. IF (CondExp1) 2. StmtExp1 3. ENDIF

대안/예외 시나리오 이름에 기입된 “-MF숫자”는 기본 시나리오의 “숫자” 번째 문장에서 시나리오가 분기되었음을 표시한다. 즉, 예제 시나리오는 기본 시나리오와 0번째 문장에서 분기하는 대안 시나리오, 1번째 문장에서 분기하는 예외 시나리오로 구성된다. 각 시나리오는 하나의 분기문을 가진다. Fig. 5는 Table 4의 예제를 이용해 구축된 테스트 모델이다.

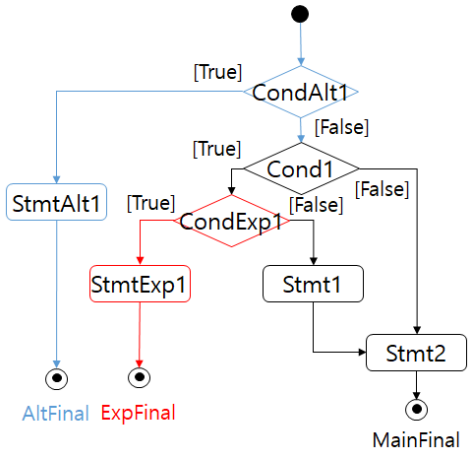


Fig. 5. Test model example

기본 시나리오를 기반으로 우선 테스트 모델을 구축한 후, 각 대안/예외 시나리오를 반영하도록 테스트 모델을 확장한다. 시나리오 명세의 단말 노드에 시나리오의 유형에 따라 각각 기본/대안/예외 Final 노드를 추가한다.

3.1.2 테스트 시퀀스 생성

테스트 시퀀스 생성은 테스트 모델의 모든 분기를 적어도 한번 수행하는 경로들을 생성하는 단계이다. DFS 알고리즘을 이용하여 경로를 탐색한다. 반복문에 의한 무한 루프를 막기 위해 각 Decision 노드의 True 분기는 최대 한번만 방문한다. 생성된 경로의 Final 노드의 이름에 따라 생성된 테스트 경로에 기본/대안/예외 속성을 부여한다. Table 5는 Fig. 5의 테스트 모델 예제를 이용해 생성한 테스트 경로이다.

Table 5. Test sequence generation example

TSMain1 : [CondAlt1=False] → [Cond1=True] → [CondExp1=False] → Stmt1 → Stmt2

TSMain2 : [CondAlt1=False] → [Cond1=False] → Stmt2

TSAlt1 : [CondAlt1=True] → StmtAlt1

TSExp1 : [CondAlt1=False] → [Cond1=True] → [CondExp1=True] → StmtExp1

3.2 테스트 데이터 분석

테스트 데이터 분석을 위해 입력 변수를 추출하고 추출된 입력 변수의 invariant 경계값을 생성한다. 그리고 각 테스트 시퀀스에서 입력 변수가 포함된 분기 노드 조건들을 조합하여 테스트 시퀀스를 수행하기 위한 입력 변수 조건을 생성한다.

입력 변수 추출은 유스케이스 이벤트 모델과 유스케이스 명세를 기반으로 유스케이스의 입력 변수를 추출하는 단계이다. 유스케이스의 입력 information flow의 source 데이터 스토어와 flow item, 그리고 유스케이스 명세의 트리거 변수가 유스케이스의 입력 변수로 추출된다. 각 변수는 System/CSCI Interface Model, Data Model에 클래스, 이벤트, 데이터 타입 형태로 정의된다. 각 변수에는 내부 속성에 대한 invariant가 정의된다.

입력 변수의 invariant 경계값 생성은 추출된 입력 변수의 invariant를 기반으로 경계값에 해당하는 데이터를 생성하는 단계이다. int, float, char 타입의 변수는 invariant의 min-, min, min+와 max-, max, max+값을 생성하는 3-value 경계값을 생성한다. structure 타입의 변수가 structure 변수를 포함한 경우, 테스터가 설정한 input 레벨까지의 하위 structure만을 경계값 생성 대상으로 한다. array 타입의 변수는 유효인덱스 범위의 경계값 뿐만 아니라 유효인덱스+1 크기를 가지는 invalid 데이터도 생성한다. enum 타입의 변수는 모든 유효 enum 값만 생성한다. boolean 타입의 변수는 true, false 값을 생성한다.

경계값 자동 생성을 위해 Constraint Solver 도구를 이용한다. 유스케이스의 입력 변수가 다수인 경우 생성된 데이터를 all/pairwise/each choice 방법을 이용해 조합한다. 테스트 데이터 생성에 사용할 데이터 조합 방법은 테스터가 선택한다.

테스트 시퀀스 입력 조건 생성은 테스트 시퀀스에서 입력 변수가 포함된 분기 노드 조건들을 조합하여 각 테스트 시퀀스가 수행되기 위해 만족해야 하는 입력 변수 조건을 생성하는 단계이다. 조건문의 조건식이 자연어로 기술되는 반정형화된 시나리오 표기법을 사용하기 때문에, 입력 변수 조건을 만족하는 값을 자동 생성하지는 않는다.

3.3 테스트 케이스 생성

테스트 케이스는 입력 변수, 입력 변수 조건, 테스트 시퀀스, 예상 출력으로 구성된다. 입력 변수는 각 입력 변수의 이름과 invariant, 그리고 invariant를 기반으로 생성한 경계값들로 구성된다. 입력 변수 조건은 각 입력 변수별로 시험 절차를 수행하기 위해 요구되는 입력 변수의 조건으로 구성된다. 테스트 시퀀스는 테스트 스텝들로 구성되는데, 각 테스트 스텝은 문장 번호와 문장 유형을 포함한다. 예상 출력은 출력 변수가 포함된 노드와 유스케이스의 post condition을 조합하여 생성된다. 유스케이스의 출력 변수는 유스케이스의 출력 information flow의 target 액터/데이터 스토어와 flow item로 추출된다.

4. 사례 적용

사례 적용에서는 사례 적용을 수행할 대상 모델을 소개하고 개발한 테스트 케이스 생성 도구를 설명한다. 그리고 사례 적용 결과를 설명한다.

4.1 대상 모델

LIG넥스원에서 개발 중인 항전 제어 시스템의 소프트웨어 요구사항을 명세한 무기체계 소프트웨어 참조 모델을 대상으로 사례 적용을 수행한다. 대상 모델은 항전 제어 시스템의 시스템 관리, 미션 관리, 항공기 관리, 인터페이스 관리 기능에 대한 세부 요구사항들에 대해 모델링되었다. Table 6은 대상 모델의 규모를 표현한다.

Table 6. Target model size

Model Name	# of Scenario	# of Step	# of Var.
Current Time Management	2	11	1
Fault Management	4	25	1
System Initialization	3	9	2
Auto Switching Mode Management	2	9	0
Flight Plan Management	2	12	2
Vehicle Fault Management	2	7	1
Vehicle Status Notification	2	5	1
Send Message Management	4	15	0

각 유스케이스 명세는 하나 이상의 대안 혹은 예외 시나리오를 가진다. 그리고 모든 시나리오는 13 이하의 단계로 구성된다. 입력 변수 수가 0개인 유스케이스는 결함 발생이나 타임아웃 같은 트리거가 구체적으로 정의되지 않고 자연어로 기술되어 자동화된 변수 식별이 불가능한 경우이다.

4.2 테스트 케이스 생성 도구

테스트 케이스 생성 도구는 Enterprise Architect 모델링 도구를 이용해 개발된 무기체계 소프트웨어 참조 모델을 입력 받아 XML 형태의 테스트 케이스를 생성하는 도구이다. Fig. 6은 테스트 케이스 생성 도구의 구조도를 표현한다.

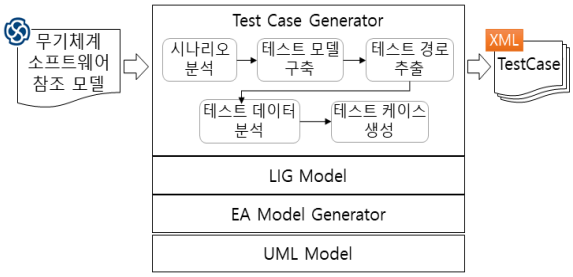


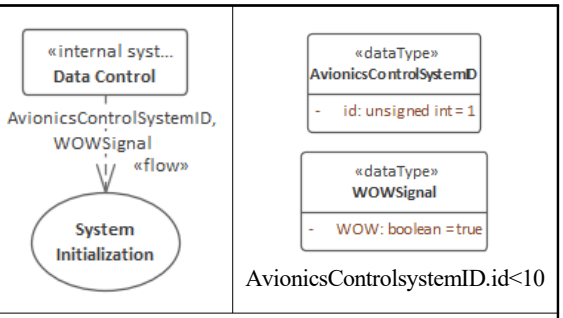
Fig. 6. Test case generation tool structure

테스트 케이스 생성 도구는 Test Case Generator 레이어, LIG Model 레이어, EA Model Generator 레이어, UML Model 레이어로 구성된다. Test Case Generator 레이어는 입력된 모델의 유스케이스 시나리오를 분석하여 테스트 모델을 구축하고 테스트 시퀀스와 테스트 데이터를 분석하여 테스트 케이스를 생성하는 기능을 수행한다. LIG Model 레이어는 무기체계 소프트웨어 참조 모델 구조를 정의하고 생성한다. EA Model Generator 레이어는 입력된 Enterprise Architect 모델 파일을 분석하여 UML Model을 생성한다. UML Model 레이어는 표준 UML 모델 구조를 정의한다. Rhapsody와 같은 다른 모델링 도구에 대한 확장이 용이하도록 UML Model 레이어와 UML Model 생성 레이어를 분리하였다.

4.3 사례 적용 결과

제안 방법을 8개의 대상 모델에 적용하여 테스트 케이스를 생성하였다. 본 절에서는 8개의 대상 모델 중 3개의 시나리오와 2개의 입력 변수를 가지는 “System Initialization”를 모델을 대상으로 생성된 테스트 케이스 생성 결과를 설명한다. Table 7은 “System Initialization”의 유스케이스 이벤트 모델, 데이터 명세, 유스케이스 명세를 표현한다.

Table 7. “System Initialization” model & use case specification



<ID>
R-ACS-SFR-SM-01-001
<definition>
1. Avionics Control System을 초기화 한다.
<triggers>
내부 $Data Control$ 시스템으로 부터의 $Avionics ControlSystemID$, $WOWSignal$ 데이터 수신
<pre-condition>
$Avionics Control System$은 PBIT을 완료해야 한다.
<main flow>
1. RECEIVE $WOWSignal$, $AvionicsControlSystemID$ FROM $Data Control$
2. IF $WOWSignal$== TRUE
3. $WOWSignal$ 값을 설정한다.
4. ENDIF
5. IF($AvionicsControlSystemID$ >= 1) && ($AvionicsControlSystemID$ < 10)
6. $AvionicsControlSystemID$ 값을 설정한다.
7. ENDIF
<alternative flow-MF2>
1. ELSE WOW 결함으로 설정한다.
<alternative flow-MF5>
1. ELSE $AvionicsControlSystemID$ 결함으로 설정한다.
<post-condition>
시스템 운용 모드가 대기모드로 설정된다.

Table 8. “System Initialization” test case generation result

```
<testsuite testID="R-ACS-SFR-SM-01-001"
 testItem="System Initialization">
  <preCondtion>
$Avionics Control System$은 PBIT을 완료해야 한다.
  </preCondtion>
  <testData id="TD1">
    <variable name="AvionicsControlSystemID.id“
invariant="AvionicsControlSystemID.id<10">9</variable>
    <variable name="WowSignal.WOW">true</variable>
  </testData>
  <testData id="TD2"> // .. // </testData>
  <testData id="TD3"> // .. // </testData>
  <main1 id="M1">
    <inputCondition name="AvionicsControlSystemID.id">
(($AvionicsControlSystemID$ >= 1) &&
($AvionicsControlSystemID$ < 10)) == true
    </inputCondition>
    <inputCondition name="WOWSignal">
($WOWSignal$== TRUE) == true</inputCondition>
    <sequence kind="s" seq="1">
RECEIVE $WOWSignal$, $AvionicsControlSystemID$
 FROM $Data Control$</sequence>
    <sequence kind="d" seq="2">
($WOWSignal$== TRUE) == true</sequence>
    <sequence kind="s" seq="3">
$WOWSignal$ 값을 설정한다.</sequence>
    <sequence kind="d" seq="5">
(($AvionicsControlSystemID$ >= 1) &&
($AvionicsControlSystemID$ < 10)) == true</sequence>
    <sequence kind="s" seq="6">
$AvionicsControlSystemID$ 값을 설정한다.</sequence>
  </main1>
  <alternative1 id="A1">
    <inputCondition name="WOWSignal">
($WOWSignal$== TRUE) == false</inputCondition>
    <sequence kind="s" seq="1">
RECEIVE $WOWSignal$, $AvionicsControlSystemID$
 FROM $Data Control$</sequence>
    <sequence kind="d" seq="2">
($WOWSignal$== TRUE) == false</sequence>
    <sequence kind="s" seq="1">
WOW 결함으로 설정한다.</sequence>
  </alternative1>
  <alternative2 id="A2"> // .. // </alternative2>
  <expectedResult>시스템 운용 모드가 대기모드로 설정
된다.</expectedResult>
</testsuite>
```

Table 9. Test case generation results

Model Name	# of TestSequence			# of Data
	M	A	E	
Current Time Management	1	1		3
Fault Management	1	3		0
System Initialization	1	2		3
Auto Switching Mode Management	1	1		0
Flight Plan Management	3	3		0
Vehicle Fault Management	2	1		0
Vehicle Status Notification	1		2	2
Send Message Management	1		6	0

“System Initialization”은 무인항공기의 임무 수행을 지원하기 위한 항공전자 장비들의 집합체를 제어/관리하는 항전 제어 시스템의 초기화 기능을 명세한 것이다. “System Initialization”은 항전 제어 시스템에 전원이 인가되어 OFF 상태에서 Active 상태로 전환되는 시점에 수행하는 자가 진단 테스트인 PBIT(Power up Built In Test)를 완료한 상태에서, 항전 제어 시스템의 식별자인 AvionicsControlsystemID와 항공기가 지상 위비 여부를 판단하는 근거 정보를 SW에서 인지하는 신호인 WOW(Weight On Wheel) Signal을 입력받아 수행된다.

“System Initialization”은 선/후행조건, 기본 시나리오와 두 개의 대안 시나리오를 가진다. 예외 시나리오는 가지지 않는다. Table 8은 Table 7 입력에 대한 테스트 케이스 생성 결과이다.

테스트 케이스 생성 결과 3개의 테스트 데이터와 메인 테스트 시퀀스 1개, 대안 테스트 시퀀스 2개를 생성하였다. 테스트 데이터는 each once 기준으로 생성하였다. Table 9는 전체 대상 모델에 대한 테스트 케이스 생성 수를 정리한 것이다.

테스트 케이스 생성 결과 8개의 대상 모델을 대상

으로 11개의 기본 테스트 시퀀스, 11개의 대안 테스트 시퀀스, 8개의 예외 테스트 시퀀스, 8개의 테스트 데이터를 생성하였다. 테스트 데이터가 생성 되지 않은 경우는 입력이 값을 생성할 수 없는 별도의 속성을 가지지 않는 시그널이거나 입력 변수가 정의되지 않은 경우 혹은 입력 변수의 invariant가 정의되지 않은 경우이다.

5. 관련 연구

Alrawashed의 연구[7]는 UML 유스케이스 모델의 유스케이스를 Cockburn의 연구[8]에서 정의한 유스케이스 템플릿을 이용하여 정의한다. 정의한 유스케이스 템플릿을 바탕으로 제어 흐름도를 자동으로 생성한다. 그리고 해당 연구에서 제안한 도구를 이용하여 테스트 시퀀스를 생성한다.

Sarmiento의 연구[9]는 자신의 연구에서 정의한 제한된 자연어를 이용하여 요구사항 시나리오를 정의한다. 그리고 정의한 시나리오들을 이용해 Petri-Net을 생성한다. 생성된 Petri-Net을 분석하여 명확성, 완전성, 일관성, 정확성을 해치는 모델 결함들을 제거하고, 최종 완성된 Petri-Net을 요구사항 엔지니어가 최종적으로 확인 후, Petri-Net을 기반으로 테스트 시퀀스를 생성한다.

Alrawashed의 연구와 Sarmiento의 연구는 Cockburn의 유스케이스 템플릿과 제한된 자연어로 요구사항만을 정의하며, 시스템에서 사용되는 데이터를 정의하지 않는다. 따라서 테스트 데이터를 생성하는 본 연구와 달리 테스트 시퀀스만 생성하며, 테스트 시퀀스에 대한 입/출력 테스트 데이터를 생성하지 않는다.

Kesserwan의 연구[10]는 UML 유스케이스 모델의 유스케이스를 Cockburn의 연구에서 정의한 유스케이스 템플릿을 이용하여 정의한다. 유스케이스 템플릿을 바탕으로 시나리오 모델을 생성하고 생성한 시나리오 모델을 기반으로 테스트 시나리오를 생성한다. 각 테스트 시나리오에서 사용되는 테스트 데이터를 위한 데이터 모델을 정의하고, 테스트 시나리오와 데이터 모델을 이용하여 테스트 케이스를 생성한다.

Kesserwan의 연구는 제한된 형태의 요구사항과 데이터 모델을 이용하여 테스트 시퀀스를 생성하고 각 스텝에 사용되는 입력 변수를 식별한다. 그러나 요구사항으로부터 테스트 모델을 생성하는 과정이 자동화된 본 연구에 비해, 요구사항-시나리오 모델-테스트 시나리오-데이터 모델 간 관계를 수동으로 매핑해야 한다는 한계점이 있다. 그리고 테스트 데이터 값을 생성하는 본 연구에 비해, 입력 변수만 식별할 뿐 구체적인 값을 생성하지 않는다.

Hue의 연구[11]는 UML 유스케이스 모델의 유스케이스를 유스케이스 명세 언어 모델[12]을 이용하여 표현하고, 이를 기반으로 유스케이스 시나리오와 시나리오를 구성하는 각 스텝의 제약사항을 생성한다. 유스케이스 명세 언어를 구성하는 각 제약사항은 제약사항 표기 언어인 Object Constraint Language(OCL)로 기술된다. 각 스텝의 제약사항과 제약사항에 기술된 입력 변수를 OCL 표현식을 풀어내는 OCL Solver를 이용하여 분석함으로써 테스트 시퀀스와 테스트 데이터를 생성한다.

Hue의 연구는 테스트 시퀀스와 각 스텝에서 사용되는 테스트 데이터의 구체적인 값을 생성한다. 그러나 자연어를 기반으로 하는 반정형화된 명세를 사용하는 본 연구에 비해, OCL과 같은 엄격한 정형 명세를 이용하여 요구사항 명세 및 모델을 정의하는 것은 실세계의 소프트웨어 개발에서 적용하기 어려운 방법이다.

6. 결 론

본 논문에서는 무기체계 소프트웨어 요구사항/설계 모델인 무기체계 소프트웨어 참조 모델을 기반으로 테스트 케이스를 자동 생성하는 방법을 제안하였다. 테스트 케이스 생성을 위해 유스케이스 명세의 메인/대안/예외 시나리오를 분석하여 테스트 시퀀스를 생성하고 유스케이스 이벤트 모델과 유스케이스 명세를 기반으로 테스트 데이터를 생성하였다. 그리고 제안 방법을 구현한 테스트 케이스 생성 도구를 이용하여 항전 제어 시스템을 구성하는 8개의 대상 모델을 입력으로 11개의 기본 테스트 시퀀스, 11개의 대안 테스트 시퀀스, 8개의 예외 테스트 시퀀스, 8개의 테스트 데이터를 생성하였다.

무기체계 소프트웨어 참조 모델을 이용하여 테스트 시퀀스와 테스트 데이터를 모두 고려한 테스트 케이스를 자동 생성하였다. 테스트 케이스 자동 생성을 통해 테스터의 역량에 의존하는 수동 테스트 케이스 생성에 비해 일관된 품질의 테스트 케이스를 생성할 수 있었다. 그리고 정형화된 시나리오 명세가 아닌 자연

어 기반의 반정형화된 시나리오 명세를 이용하여 테스트 케이스를 생성함으로써 제안 기술의 적용성을 높였다. 또한 요구사항 시나리오의 분기를 모두 커버하는 시퀀스를 분석하여 세분화된 테스트 케이스를 생성함으로써 요구사항 커버리지만을 만족하는 테스트 케이스가 아닌, 다양한 시나리오 동작 흐름을 테스트할 수 있는 테스트 케이스를 생성하였다. 무기체계 소프트웨어의 유스케이스 이벤트 모델을 기반으로 요구사항과 연관된 데이터와 invariant를 추출하여 분석함으로써 테스트 시퀀스 뿐만 아니라 테스트 데이터를 함께 고려한 테스트 케이스를 생성하였다. 테스트 케이스 자동 생성을 통해 테스트 케이스 생성 비용과 시간을 절감할 수 있을 것으로 기대된다.

후 기

이 과제는 부산대학교 기본연구지원사업(2년)에 의하여 연구되었음.

References

[1] B. Beizer, “Software Testing Techniques,” Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[2] M. Xiao, M. El-Attar, M. Reformat, “Empirical Evaluation of Optimization Algorithms When Used in Goal-Oriented Automated Test Data Generation Techniques,” Empirical Software Engineering, Vol. 12, No. 2, pp. 183-239, 2007.

[3] Defense Acquisition Program Administration, STD : Software Test Description, 2017.

[4] D. H. Kim, and Y. H. Kim, C. S. Kim, “Requirements Analysis and Specification for Enhancing Reusability,” Journal of the Korean Association of Defense Industry Studies, Vol. 18, No. 1, pp. 1-17, 2011.

[5] D. H. Kim, and S. Y. Lee, B. Y. Lee, B. H. Park, “A Study on Reliable Documentation for Weapon System Software,” Journal of KIISE, Vol. 35, No. 12, pp. 61-68, 2017.

[6] H. J. Choi, and Y. W. Lee, J. H. Lee, J. S. Baek, D. H. Kim, K. T. Cho, H. S. Chae, “Development of Reference Model and Quality Measurement Tool for Quality Evaluation of Weapon System Software,” Journal of KIISE, Vol. 25, No. 03, pp. 179-190, 2019.

[7] T. A. Alrawashed, A. Almomani, A. Althunibat, and A. Tamimi, “An Automated Approach to Generate Test Cases from Use Case Description Model,” Computer Modeling in Engineering and Sciences, Vol. 119, No. 3, pp. 409-425, 2019.

[8] A. Cockburn, “Structuring Use Cases with Goals,” Journal of Object-Oriented Programming, Vol. 10, No. 5, pp. 56-62, 1997.

[9] E. Sarmiento, J. C. S. P. Leite, E. Almentero, and G. S. Alzamora, “Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets,” Electronic Notes in Theoretical Computer Science, Vol. 329, pp. 123-148, 2016.

[10] N. Kesserwan, R. Dssouli, J. Bentahar, B. Stepien, and P. Labreche, “From Use Case Maps to Executable Test Procedures: A Scenario-based Approach,” Software and Systems Modeling, Vol. 18, No. 2, pp. 1543-1570, 2019.

[11] C. T. M. Hue, D. H. Dang, N. N. Binh, and A. H. Truong, “USLTG: Test Case Automatic Generation by Transforming Use Cases,” International Journal of Software Engineering and Knowledge Engineering, Vol. 29, No. 9, pp. 1313-1345, 2019.