정규논문 (Regular Paper)
방송공학회논문지 제17권 제3호, 2012년 5월 (JBE Vol. 17, No. 3, May 2012)
http://dx.doi.org/10.5909/JBE.2012.17.3.529

방송 수신 소프트웨어의 사용자 요구 품질 향상이 가능한 예외상황 테스트케이스 자동생성 기법

최 인 화[a], 조 민 주[a], 백 종 호[a]‡, 황 준[a]

Automatic Generation Method of Exceptional Test Cases for improving User Requirement Quality on Broadcast Receiver Software

Inhwa Choi[a], Minju Cho[a], Jongho Paik[a]‡, and Jun Hwang[a]

요 약

소프트웨어 생성 주기 동안에 품질을 제어할 수 있는 매우 중요한 영역 중의 하나로 소프트웨어 테스트 방안을 들 수 있다. 일반적으로 소프트웨어 테스트케이스는 사용자 요구에 대한 이해를 기반으로 생성되지만, 사용자가 요구하는 수준의 품질을 충족시킬 수 있는 측정 가능한 테스트케이스를 생성하는 일은 결코 쉬운 일이 아니다. 특히 비기능적 요소나 예외상황에 대한 테스트 케이스 생성은 테스터의 경험에 많이 의존하기 때문에 매우 어려운 부분이다. 본 논문에서는 이러한 문제를 해결하기위한 방안으로 방송 수신 소프트웨어의 품질 측정 시 예외상황에서 발생 가능한 테스트케이스를 자동으로 생성할 수 있는 기법을 제안한다. 제안된 기법의 우수성을 검증하기 위해 상용 방송 수신 소프트웨어를 이용하여 기존 기법과의 비교 테스트를 수행하였다. 모의실험을 통해 본 논문에서 제안한 기법을 적용하여 다양한 예외상황에서 자동 생성된 테스트케이스를 수행한 결과로 기존 기법을 적용한 경우와 비교하여 7.08%의 결함을 더 발견할 수 있었다.

Abstract

Testing is an important part of quality control in the software life cycle. One of the most important issues in the software testing is to generate the appropriate test cases. Generally, the software can be tested based on product understanding. However, it is not easy to create test cases that can ensure the quality of the software according to the clients' request. Especially, it is difficult to create test cases for abnormal or exceptional situations. In this paper, we present a novel approach to generate exceptional test cases at the design level of UML model. Furthermore, we describe the results of actual experiment where DAB(Digital Audio Broadcasting) parsing program is tested with previous method and also with the proposed method. The results implies that our proposed method of generating test cases for exceptional situations detect more faults than that of previous method by 7.08%.

Keyword : exceptional test case, generating test cases, testing, DAB test

a) 서울여자대학교 멀티미디어학과 (Department of Multimedia, Seoul Women's University)
‡ 교신저자 : 백종호 (Paik, Jong Ho)
E-mail: paikjh@swu.ac.kr
Tel: +82-2-970-5606, Fax: +82-2-970-5981
※이 논문은 2012학년도 서울여자대학교 컴퓨터과학연구소 교내학술연구비의 지원을 받았음.
・접수일(2012년5월2일),수정일(2012년5월25일),게재확정일(2012년5월25일)

Ⅰ. 서 론

소프트웨어 테스트는 소프트웨어 개발 프로세스의 각 단계에서 수행되어지는 작업의 품질을 측정하고 평가하기 위한 행위로서 소프트웨어가 계획된 대로 수행되는지를 확인하여 소프트웨어의 품질과 신뢰성 그리고 유지보수성을 향상시키는데 목적이 있다. 고품질의 소프트웨어를 보장하기 위해서 다양한 테스트 기법이나 자동화 등이 이슈가 되고 있는데 특히 테스트케이스 자동생성은 중요한 이슈 중에 하나이며, 많은 연구들이 설계 단계의 UML (Unified Modeling Language) 다이어그램을 사용해 테스트케이스를 생성하는 기법들을 제안하였다[1][2][3][4][5][6][7].

일반적으로 테스트케이스는 도메인에 대한 이해를 기반으로 생성된다. 하지만 사용자의 요구사항을 100% 이해하기란 결코 쉬운 일이 아니다. 게다가 시스템이 예외상황을 만났거나 동시에 발생하는 조건들에 대해 서로 다른 응답을 하는 경우에 대한 테스트케이스를 생성하는 것은 매우 어려운 영역이다[8]. 그러나 대부분의 연구에서 예외상황에 대한 테스트케이스 생성을 다루지 않고 있지 않다. 일부 제안된 예외상황 테스트 케이스 생성에 관한 연구에서도 대부분이 예외적인 데이터에 집중하고 있으며, 수행 절차나 시간에 대한 예외상황은 전혀 고려되지 않고 있는 실정이다.

따라서 본 논문에서는 수행 시간과 절차에 초점을 두고 예외상황에 대한 테스트케이스를 자동 생성하는 기법을 제안한다. 제안하는 접근기법을 적용하기위해 디지털오디오 방송(Digital Audio Broadcasting) 수신 시스템의 어나운스먼트 기능을 대상으로 설명한다. 제안하는 기법은 세 가지 단계로 설명한다. 먼저 기능을 UML의 액티비티 다이어그램으로 표현한다. 그리고 테스트 시나리오와 케이스를 자동생성하기 위해 그래프 표현식을 정의해 다이어그램을 그래프로 나타낸다. 마지막으로 제안하는 알고리즘을 이용해 예외상황에 대한 테스트케이스를 생성한다. 제안한 기법을 검증하기 위해 상용 소프트웨어를 대상으로 제안하는 기법과 기존 기법을 이용해 각각 테스트를 수행하여 비교 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 예외상황 테스트케이스 자동생성 기법에 대해 기술하고, 3장에서는 제안한 기법을 적용해 테스트를 수행한 결과를 제시한다. 그리고 4장에서 결론을 맺는다.

Ⅱ. 예외상황 테스트케이스 자동생성 기법

소프트웨어의 예외상황은 외부요인, 오작동, 프로그램상의 오류 등 다양한 상황에서 나타날 수 있다. 따라서 예외상황에 대한 테스트 케이스를 생성하려면, 케이스 작성자의 많은 수행경험이 필요하다. 하지만 테스트 수행자가 도메인 전문가인 경우가 극히 드물기 때문에, 도메인에 관련된 케이스 보다는 데이터 값에 대한 예외처리만 진행되는 경우가 많다. 이러한 문제를 해결하기 위해 본 논문에서 액티비티 다이어그램을 이용해 정상적인 기능수행 시간 이외의 시간에 예외의 상황을 도출하는 기법을 제안한다. 예외 상황 도출을 위해 먼저 1절과 2절에서 일반적인 기법을 이용해 테스트케이스 도출하는 과정을 설명한다. 그리고 3절에서 본 논문에서 제안한 예외상황 도출 기법을 설명한다. 테스트 케이스 도출기법은 아래와 같다.

① 정상적인 기능수행 시간 정의
정상적인 기능수행 시간은, 특정 기능이 조건을 만족하여 수행을 시작하는 시간부터 수행을 끝마치는 시간까지로 한다. 이후로, 정상적인 기능 수행 시간을 'ET(Execu- tion Time)'이라 칭한다.

② 1을 기준으로 예외상황을 발생시킬 수 있는 시간 정의
예외상황을 발생시킬 수 있는 시간은 정상적인 기능수행 시간 이외의 시간 즉, 이전, 이후 그리고 조건이 만족되지 못한 시간(정상적인 기능수행 시간인데, 조건이 맞지 않아 수행할 수 없는 시간)으로 정의한다.

③ 정의된 예외상황 발생 가능 시간 내에서 예외상황을 발생시킬 주체 선정
정의된 모든 기능은 액티비티 다이어그램으로 표현할 수 있다. 다이어그램은 기능 수행에 필요한 여러 앤티

티로 구성되는데, 각 앤티티마다 고유의 상태를 갖고 있다. 일반적으로 메시지 기반의 모바일 방송 소프트웨어에서 정의되는 기능의 상태는 크게 세 가지, 메시지를 받기 위해 대기 중인 “Waiting”상태, 메시지를 받고 있는 “Receiving”상태 그리고 명령을 내리는 “Commanding”상태로 구분할 수 있다. 이들 중에서 예외상황을 발생시킬 수 있는 상황은 다른 작업을 수행하지 않는 상태이어야 하기 때문에, “Receiving”상태나 “Commanding”상태는 불가능하고 “Waiting”상태로 정의한다.

④ 예외상황 발생기법 정의

예외 상황을 자동으로 발생하기 위해서는 먼저, 액티비티 다이어그램을 도출하고 이를 그래프로 표현해야 한다. 그 후, 생성된 그래프를 기반으로 ETC(Exceptional Test Case)생성 알고리즘을 이용해 예외 테스트 케이스를 생성한다. 자세한 내용은 이후에 기술한다.

1. 액티비티 다이어그램 설계

본 논문에서 제안한 기법을 설명하기 위해 DAB(Digital Audio Broadcasting) System의 어나운스먼트 기능을 예로 든다. 어나운스먼트는 다양한 라디오 방송을 듣고 있는 사용자들에게 뉴스나 날씨 등의 공지로 알릴만한 사항을 전달할 수 있는 기능이다. 하지만 모든 라디오 방송이 이 기능을 지원하는 것은 아니다. 따라서 사용자가 이 기능을 지원하는 라디오방송을 듣고 있을 경우에만 해당 정보(announcement information)를 단말에 알린다. 이 정보를 받은 단말은 사용자에게 어나운스먼트 서비스가 존재함을 알리고, 사용자가 듣기를 원할 경우, 듣고 있던 라디오방송은 잠시 멈추고 어나운스먼트 서비스를 제공한다. 이 서비스가 종료되면 단말은 종료정보(announcement end information)를 전달 받고, 즉시 이전에 듣던 라디오 방송을 사용자에게 제공한다. 이때, 종료정보는 어나운스먼트 서비스를 지원하지 않는 라디오 서비스를 듣고 있는 사용자의 단말에도 전달된다.

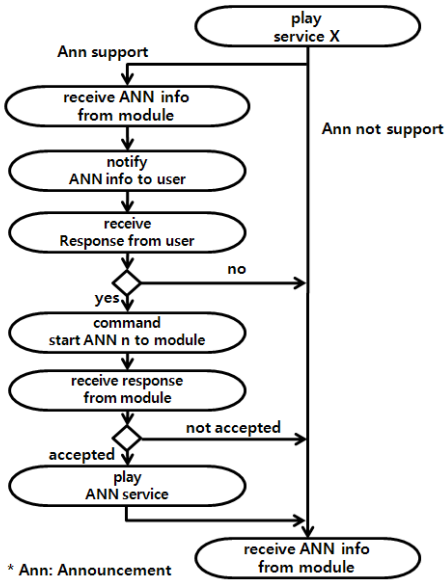

그림 1. 어나운스먼트 기능 Flow
Fig. 1. Flow of Announcement Function

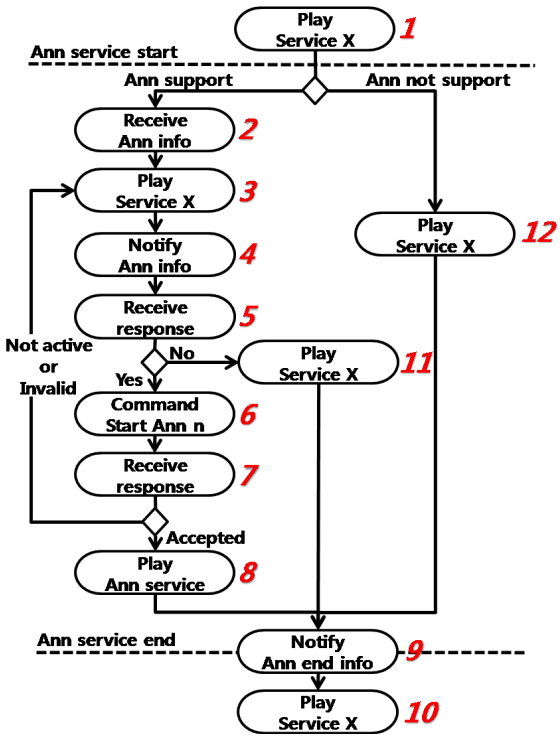

그림 2. 어나운스먼트 기능 액티비티 다이어그램
Fig. 2. Activity Diagram of Announcement Function

그림 2는 어나운스먼트 기능을 표현한 액티비티 다이어그램을 나타낸다. 그림 1의 다이어그램에는 위에서 기술했던 어나운스먼트 기능이 모두 표현되었음을 볼 수 있다. 하지만 이 다이어그램으로는 각 앤티티의 상태가 표현되지 않기 때문에, 수행시간을 기준으로 도출하는 테스트 케이스를 생성하는데 많은 어려움이 따른다. 따라서 그림 2와 같이 상태가 변하는 앤티티의 상태를 적용하여 액티비티 다이어그램을 다시 표현하였다.

표 1. 테스트 시나리오
Table 1. Test Scenario

Test Scenario	Description
1→12→9→10	어나운스먼트 서비스를 지원하지 않는 경우
1→2→3→4→5→11→9→10	어나운스먼트 서비스를 지원하지만 User가 원치 않아서 어나운스먼트 서비스를 무시하는 경우
1→2→3→4→5→6→7→8→9→10	어나운스먼트 서비스를 지원하고, User가 원하여 어나운스먼트 서비스를 제공받는 경우
1→2→3→4→5→6→7→3→4→5→6→7→8→9→10	어나운스먼트 서비스를 지원하고 User가 원하여 실제로 어나운스먼트 서비스를 수행할 때 오류가 발생, 재요청 의사를 물어 어나운스먼트 서비스를 제공받는 경우
1→2→3→4→5→6→7→3→4→5→11→9→10	어나운스먼트 서비스를 지원하고 User가 원하여 실제로 어나운스먼트 서비스를 수행할 때 오류가 발생, 재요청 의사를 물을 때 User가 거부하는 경우

그림 2를 기반으로 수행할 수 있는 테스트 케이스 시나리오는 다섯 가지로, 표 1과 같이 나타낼 수 있다.

2. 액티비티 다이어그램 앤티티 상태 정의 및 그래프 표현식 정의

테스트의 자동 수행과 예외상황 테스트케이스의 도출을 위해서 액티비티 다이어그램을 그래프로 표현해야 한다. 그래프 G는 액티비티 다이어그램의 앤티티를 나타내는 노드 N과 앤티티들 간의 관계를 나타내는 조건 C로 표현하며, N은 엣지를 나타내는 E와 상태를 나타내는 S로 구성된다.

$$G = \{N, C\} \tag{1}$$

$$N = \{E, S\} \tag{2}$$

$$E = \{N_{strt}, N_{dest}\} \tag{3}$$

$$S = Waiting / Receiving / Commanding \tag{4}$$

E는 N의 연결정보로써, 시작 노드를 나타내는 Ndest과 목적 노드를 나타내는 Ndest를으로 구성된다. S는 N의 세 가지 상태, Waiting, Receiving, Commanding을 표현하며, 예외상황 테스트케이스를 생성에 중요한 요소로 사용된다.

$$C = \{E, C_{level}, FT\} \tag{5}$$

$$C_{level} = 0/1 \tag{6}$$

$$ET = T_n / T_b / T_a / T_u \tag{7}$$

C는 하나의 N에 여러 개의 E가 존재할 경우, E별로 분기할 수 있는 모든 경우를 나타내는 정보이다. 예외상황을 발생시킬 때에도 이 C정보를 이용해 특정 E로 분기한다. (5)에서 C의 요소 E는 N에 속한 E들 중 하나를 나타낸다. (6)에 나타난 Clevel은 수행조건에 따라 수행이 가능한지의 여부를 나타내는 값으로, 0일 경우는 해당 조건을 수행하지 않고, 1일 경우에 수행한다. (7)의 T는 네 가지 경우의 수행시간을 나타낸다. 첫 번째는 정상적인 수행시간으로 Tn으로 표기한다. 두 번째는 약속된 수행시간 이전의 수행시간으로 Tb로 표기한다. 세 번째는 약속된 수행시간 이후의 시간으로 Ta로 표기하며 마지막으로 조건에 맞지 않는 시간으로 Te를 나타낸다.

정의된 N정보를 할당하기 위해서 일련의 함수가 정의되어야 한다. 아래 대표적인 함수들을 소개한다.

$$AllocEdge(N_{id}, E_{id}) \tag{8}$$

$$AllocStatus(N_{id}, Status) \tag{9}$$

$$CreateEdge(E_{id}, N_{id}, N_{id}) \tag{10}$$

$$AllocConn(E_{id}, C_{flag}, ET) \tag{11}$$

표 2. 그래프 정보
Tabel 2. Graph Information

Edge Information	
E1=(N1, N2) E2=(N2, N3) E3=(N3, N4) E4=(N4, N5) E5=(N5, N6) E6=(N6, N8) E7=(N7, N8) E8=(N8, N9) E9=(N9, N10) E10=(N5, N11) E11=(N11, N9) E12 =(N7, N3) E13 =(N1, N12) E14 =(N12, N9)	
Node & Connection Information	
N1 = {(E1, E13), "WAITING", C1} N2 = {(E2), "RECEIVING", C2} N3 = {(E3), "WAITING", C3} N4 = {(E4), "RECEIVING", C4} N5 = {(E5, E10), "RECEIVING", C5} N6 = {(E6), "COMMANDING", C6} N7 = {(E7, E12), "RECEIVING", C7} N8 = {(E8), "WAITING", C8} N9 = {(E9), "RECEIVING", C9} N10 = {(), "WAITING", C10} N11 = {(E11}, "WAITING", C11} N12 = {(E14), "WAITING" C14}	C1 = (E1, 1, Tn), (E13, 0, Tn) / (E1, 0, Tn), (E13, 1, Tn) C2 = (E2, 1, Tn) C3 = (E3, 1, Tn) C4 = (E4, 1, Tn) C5 = (E5, 1, Tn), (E10, 0, Tn) / (E5, 0, Tn), (E10, 1, Tn) C6 = (E6, 1, Tn) C7 = (E7, 1, Tn), (E12, 0, Tn) / (E7, 0, Tn), (E12, 1, Tn) C8 = (E9, 1, Tn) C9 = (E9, 1, Tn) C10 = (END, 1, Tn) C11 = (E11, 1, Tn) C14 = (E14, 1, Tn)

함수 (8)은 특정노드 Nid 에 연결정보 Eid를 할당하기 위한 함수이다. 표 2의 Node & Connection Information에서 각 노드별로 할당된 1개 이상의 연결정보 E가 할당된 것을 볼 수 있는데 함수(8)을 이용해 할당된 정보들이다. 함수 (9)는 하나의 노드에 상태정보를 할당하는 함수이다. 위와 마찬가지로 표2에서 각 노드별로 할당된 "WAITING", "RECEIVING", "COMMANDING" 상태가 이 함수를 통해 할당된 정보들이다. 함수 (9)는 새로운 연결정보 E를 생성할 때 사용되는 함수로, 생성하고자 하는 연결정보 E가 첫 번째 파라미터이며, 시작 노드가 두 번째 파라미터, 도착 노드가 세 번째 파라미터 정보로 사용된다. 함수 (11)은 조건을 할당하는 함수이다. 첫 번째 파라미터는 조건이 주어질 연결정보 E를 나타내며, 두 번째 파라미터는 수행가능 여부를 나타내는 항목이다. 그리고 세 번째 파라미터는 수행시간 ET를 나타낸다. 함수 수행결과 표 2의 Node & Connection Information의 조건정보가 생성된다.

그림 3은 표 2의 노드 정보를 그래프 형식으로 나타낸 그림이다.

그림 2는 요구사항 정의서를 기반으로 어나운스먼트 기능을 표현한 액티비티 다이어그램 이고, 그림 3은 액티비티 다이어그램을 이용해 어나운스먼트 기능을 표현한 그래프인데, 동일하게 표현되었음을 확인할 수 있다.

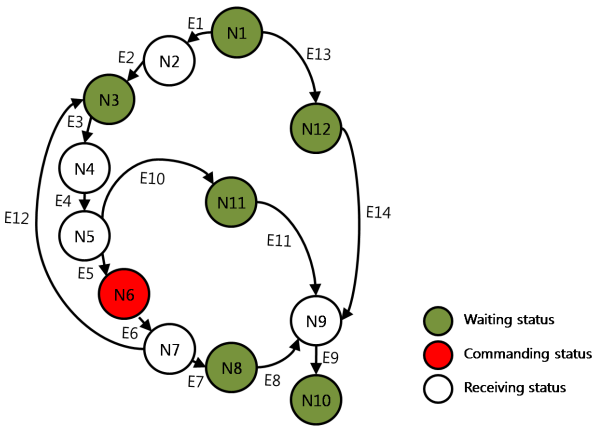

그림 3. 어나운스먼트 기능을 나타낸 그래프
Fig. 3. Graph of Announcement Function

3. 예외 테스트케이스 생성

이 부분까지는 기존기법과 유사하게 진행된다. 하지만 이 기법만으로는 돌발적으로 나타나는 예외상황을 충분히 도출하기가 어렵다. 도입부에서 설명했듯이, 2와 3의 과정을 통해 생성된 그래프 정보를 이용하여 수행시간(ET)을 기준으로 예외 테스트케이스를 생성한다. 아래 표 3에 예외 케이스를 생성하는 기준이 나타난다. 표3의 X축은 파라메터를 나타내는 변수로써, 데이터의 범위를 기준으로 테스

트 케이스를 생성하는 예이다. Y축은 ET을 나타낸다.

표 3. 예외 케이스 생성 기준
Table 3. Generation Metrics of Exceptional Case

	P_l	P_n	P_o	P_{ll}	P_{lr}
ET_b	•	√	•	•	•
ET_n	√(X)	※	√(X)	√(X)	√(X)
ET_a	•	√	•	•	•
ET_e	•	√	•	•	•

- Pl(Parameterless): 정의된 파라메터 영역보다 적은 값
- Pn(Parameternormal): 정의된 파라메터 영역에 속하는 값
- Po(Parameterover): 정의된 파라메터 영역보다 많은 값
- Pll(Parameterlimit_left): 정의된 파라메터 영역의 왼쪽 경계 값
- Plr(Parameterlimit_right): 정의된 파라메터 영역의 오른쪽 경계 값

- ETb(ExecutionTimebefore): 정의된 수행시간 이전에 수행
- ETn(ExecutionTimenormal): 정의된 수행시간에 수행
- ETa(ExecutionTimeafter): 정의된 수행시간 이후에 수행
- ETe(ExecutionTimeexecptional): 정의된 수행조건에서 벗어난 영역에서 수행

표 3 에서 ※ 기호로 표현된 케이스는 정상적인 수행시간에 정상적인 파라메터로 테스트를 수행하는 경우이다. 그 외의 경우에서는 모두 예외상황을 도출할 수 있다. 하지만 논문에서 다루는 테스트 케이스 생성 요건이 정상적이지 않은 수행시간 기준이기 때문에, √(x) 기호로 표시된 ETn 에 속하는 항목은 제외한다. • 으로 표현된 부분은 정상적이지 않은 수행시간의 범위에 속하지만 파라메터와의 연관성이 적어 Pn 에서 수행하는 경우와 동일하기 때문에 다루지 않는다.

ET는 그림 3에서 도출된 그래프의 commanding status 노드(N6)를 중심으로 구분된다.

그림 4에서 Tb, Ta, Tu 로 나타내진 영역이 ET를 나타내는 영역이다. 모든 영역에서 예외 테스트케이스를 도출하기 위해서는

①"Waiting" status 노드를 찾는다.
② 찾아진 노드가 commanding status 노드로 연결되는 E를 갖고 있는지를 확인한다.
③ 만약 E가 없다면, 찾아진 노드와 Commanding status 노드로의 E 정보를 추가한다.

$CreateEdge(E15, N1, N6)$,
$CreateEdge(E16, N3, N6)$,
$CreateEdge(E17, N11, N6)$

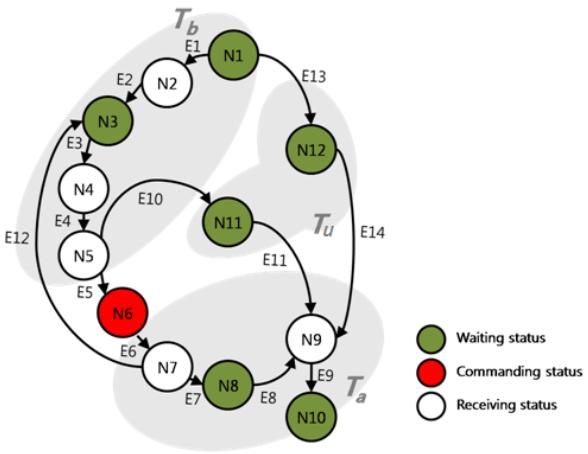

그림 4. Exceptional Time을 나타낸 그래프
Fig. 4. Graph of Exceptional Time

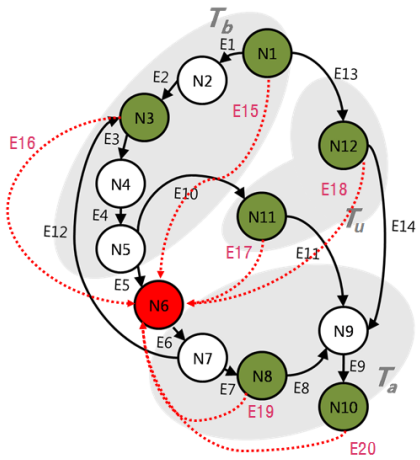

그림 5. 예외상황을 나타낸 그래프
Fig. 5. Graph of Exceptional Condition

$CreateEdge(E18, N12, N6)$,
$CreateEdge(E19, N8, N6)$,
$CreateEdge(E20, N10, N6)$

④ E 정보가 추가된 노드의 C 정보를 추가한다.

$AllocCon(E15, 1, T_b)$, $AllocCon(E16, 1, T_b)$,
$AllocCon(E17, 1, T_e)$
$AllocCon(E18, 1, T_e)$, $AllocCon(E19, 1, T_a)$,
$AllocCon(E20, 1, T_a)$

여기서, 범위 Ta에 해당되는 N은 start 노드부터 commanding status 노드까지 수행했던 노드들이고, Ta에는 commanding status 노드부터 end 노드까지 수행되었던 노드들이 해당된다. Tu 영역은 commanding status 노드와 직접적인 관련이 없는 노드이기 때문에, 이외의 수행 노드들이 이 범위에 해당된다.

그림 5는 정의된 함수를 이용해 예외상황의 테스트 케이스를 생성한 결과를 그래프로 나타낸 그림이다. 그림에서

표 4. 테스트케이스 생성 알고리즘
Table 4. Generation Algorithm of Test Cases

```
/* get node_id of the 'start node' */
start_nodeId = get_start_nodeId();
/* get node_id of the 'command node' */
cmd_nodeId = get_cmd_nodeId();
/*get number of node belong to execution time area tb */
tb_num=get_nOfNode(tb);

/* generate exceptional test cases belonging to execution time area 'Tb'*/
node_idx = start_nodeId;
for(i=0; i<get_nOfNode(tb); i++)
{
    current_node=get_node(node_idx);
     /* if node_id is node_idx, get node information */
    if((current_node.status=="WAITING")&&(!has_com_edge(current_node.id) ))
    {
        AllocEdge(get_next_edgeId(),current_node_id, cmd_nodeId);
         /* 1:exist dependency,  1:Tb */
        AllocCon(get_cur_node_d(), 1, 1);
    }
}

/* generate exceptional test cases belonging to execution time area 'Ta'*/
node_idx=get_next_nodeId(command_node);
/* get node_id of the 'end node' */
end_node = get_end_nodeId();
for(i=0; i<get_nOfNode(Ta);i++)
{
    current_node=get_node(node_idx);
    if(current_node.status=="WAITING")
    {
        AllocEdge(get_next_edgeId(), current_node, command_node);
         /* 1:exist dependency,  2:Ta */
        AllocCon(get_cur_node_d(), 1, 2);
    }
}

/* generate exceptional test cases belonging to execution time area 'Te'*/
node_idx=start_node;
for(i=0; i<get_nOfNode(Tn); i++)
{
    if(has_com_edge(node_idx)!=1)
    {
        current_node=get_node(node_idx);
        if(current_node.status="WAITING")
        {
            AllocEdge(get_next_edgeId(), current_node.id, command_node);
              /* 1:exist dependency,  2:Ta */
            AllocCon(get_cur_node_d(), 1, 2);
        }
    }
}

// get_node(node_idx)함수는 node_idx에 해당하는 node정보를 가져온다.
// has_com_edge()함수는 current node와 command node사이의 edge 존재여부를 return한다.
// get_next_edgeId() 함수는 다음에 할당할 edge_id를 return한다.
// get_cur_node_d()는 현재 node의 dependency id를 return한다.
```

붉은 점선으로 표시된 E15~E19 가 새로이 추가된 예외상황의 테스트 케이스를 나타낸다.

표 4는 테스트케이스 생성 알고리즘 구현 예를 나타낸다.

III. 실험 결과

본 논문에서 제안하는 기법을 검증하기 위해 상용제품인 DAB의 수신 소프트웨어를 대상으로 테스트를 수행하였다. 그리고 테스트 결과의 비교를 위해 제안하는 기법을 적용하지 않은 이전모델과 제안하는 기법을 적용한 제안모델의 두 경우에 대해 실험을 진행하였다. 평가대상 항목은 각 경우에서 생성한 테스트 케이스의 수와 발견한 결함의 수로 하였다. 테스트 수행시간이 길어지고 테스트 케이스의 수가 많아지면 발견한 결함의 수가 많아지는 것은 당연하기 때문에, 본 논문에서는 두 기법에서 도출된 테스트 케이스의 수와 결함의 수의 비율을 측정하여 이를 평가항목으로 선정하였다. 테스트 수행시간은 미미한 차이를 보였기 때문에 적용하지 않았다.

아래 표 5는 명령 메시지를 기준으로 생성된 각 액티비티 다이어그램영역에서 산출된 테스트 케이스의 수(TC)와 결함의 수(FT)를 보여준다.

표 5에 나타난 결과를 살펴보면, 전체적으로 제안한 모델의 테스트케이스의 수가 이전 모델보다 많은 것을 볼 수

표 5. 이전 모델과 제안하는 모델에서 산출된 테스트케이스 수와 결함의 수
Table 5. Comparison of Test Case and Fault Between Existing Model and Proposed Model

idx	command message	Previous model(TC⇒TF)	Proposed model(TC⇒TF)
1	FACTORY	12 ⇒ 0	16 ⇒ 0
2	CLR_INVALID	30 ⇒ 3	47 ⇒ 15
3	CHDB_INFO	16 ⇒ 4	25 ⇒ 5
4	SCAN	32 ⇒ 4	36 ⇒ 4
5	SET_FREQS	9 ⇒ 0	9 ⇒ 0
6	SEEK	35 ⇒ 7	42 ⇒ 8
7	PLAY_AUDIO	6 ⇒ 0	15 ⇒ 3
8	CUR_AUDIO	4 ⇒ 0	6 ⇒ 2
9	DRC1	4 ⇒ 0	4 ⇒ 0
10	MUTE	6 ⇒ 0	6 ⇒ 0
11	ENSEMBLE	35 ⇒ 7	38 ⇒ 9
12	SERVICE	34 ⇒ 5	40 ⇒ 5
13	ANNOUNCEMENT	22 ⇒ 3	41 ⇒ 20
14	SRV_FOLLOW	50 ⇒ 12	68 ⇒ 19
15	SET_PRESET	14 ⇒ 5	16 ⇒ 6
16	PLAY_PRESET	6 ⇒ 0	10 ⇒ 0
17	GET_PRESET_LABEL	12 ⇒ 3	16 ⇒ 5
18	SET_FAVORITE	14 ⇒ 5	16 ⇒ 7
19	GET_FAVORITE	12 ⇒ 3	15 ⇒ 5
20	PLAY_FAVORITE	6 ⇒ 0	10 ⇒ 2
21	GET__FAVORIET_L	12 ⇒ 5	16 ⇒ 6

있다. 7,8,20의 경우에는 이전 모델에서는 결함을 찾지 못했지만 제안한 기법을 사용했을 때, 결함이 발견되었음을 볼 수 있다. 2,13,14의 경우는 앤티티의 상태에 민감한 기능이기 때문에, 이전 모델보다 월등히 많은 수의 결함이 발견되었다.

위의 결과만으로 볼 때는 제안한 모델이 훨씬 더 많은 결함을 발견한 것으로 보이지만, 테스트 케이스의 수가 증가하면 그만큼 결함율이 발견될 가능성도 커질 수 있다. 따라서 본 논문에서는 증가한 테스트 케이스의 수를 고려하여 발견된 결함의 비율을 측정해보았다.

각 명령 메시지에 대해
TC_pr: 이전 모델에서 산출된 테스트케이스의 수
TC_pp: 제안한 모델에서 산출된 테스트케이스의 수
FT_pr: 이전 모델에서 산출된 결함의 수
FT_pp: 제안한 모델에서 산출된 결함의 수라고 정의할 때, 이전 모델에서 발견한 결함의 비율은

$$\sum_{i=1}^{n} \frac{TC\_pr_i}{FT\_pr_i} \times 100 \quad (12)$$

제안한 모델에서 발견한 결함의 비율은

$$\sum_{i=1}^{n} \frac{TC\_pp_i}{FT\_pp_i} \times 100 \quad (13)$$

정의한 식은 각 모델에서 생성된 테스트 케이스의 수와 결함의 수를 나누어 백분율로 계산하여 테스트 케이스 생성대비 결함 발견율을 구하는 식이다. 식 (12)에 의해 14.43%, (13)에 의해 21.51%가 산출되었다. 그림 8은 각 메시지에 대해 이전모델과 제안 모델에서의 결함 발견율을 나타낸다.

그림 6은 테스트를 수행 결과를 나타낸다. x축은 측정한 기능을 나타내고, y축은 결함이 생성된 수를 나타낸다. 하나의 기능 당 두 개의 데이터가 표현되는데, 왼쪽에 위치한 짙은 색의 데이터는 기존모델을 적용했을 경우에 발견된

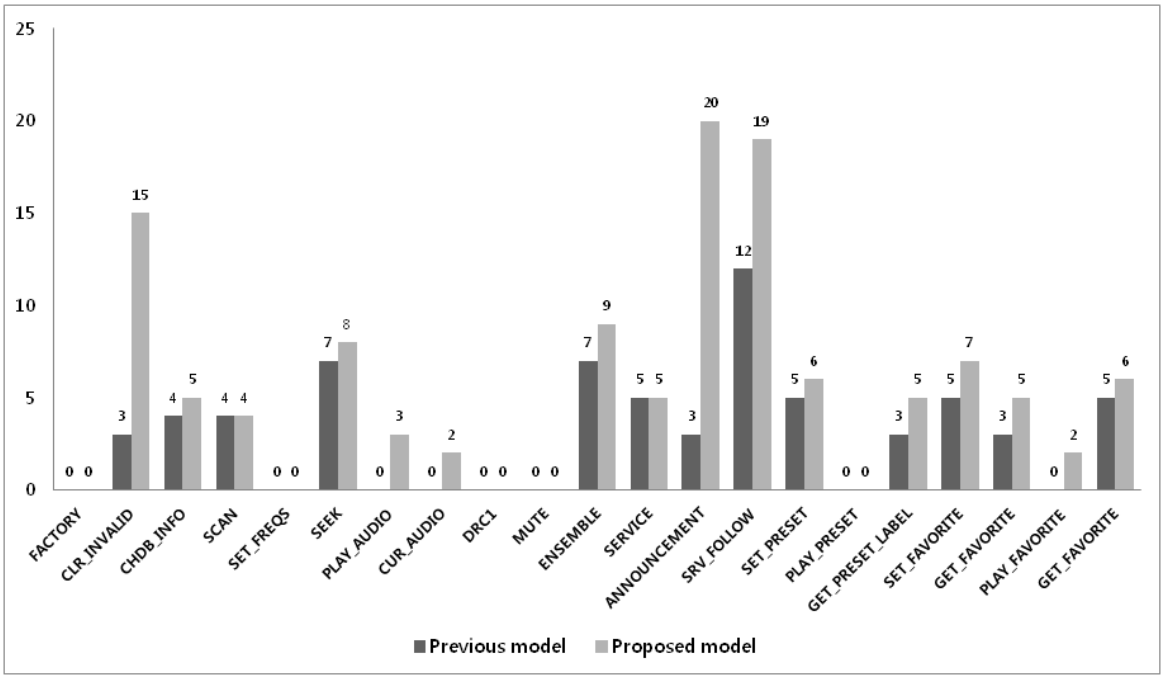

그림 6. 이전 모델과 제안 모델에서의 결함수 비교
Fig. 6. Comparison of Fault Between Existing Model and Proposed Model

결함을 나타내고 오른쪽에 위치한 옅은 색의 데이터는 제안한 모델을 적용했을 경우에 발견된 결함을 나타낸다.

이와 같은 결과를 통해 제안한 기법 즉, 수행시간을 기준으로 예외의 테스트 케이스를 도출하는 기법을 사용했을 경우, 전체적으로 기존 기법보다 7.08%의 결함 발견율이 향상됨을 알 수 있었다.

Ⅳ. 결 론

본 논문에서는 방송 수신 소프트웨어 검증 시 예외상황에서 발생 가능한 테스트케이스를 자동으로 생성하는 기법을 제안하였다. 예외상황은 시간을 기준으로 정의되는데, 사용자가 소프트웨어에게 특정 기능에 대한 요청을 할 때 약속된 시간이 아닌 예외의 시간에 기능 수행요청을 하는 상황을 말한다. 이러한 예외상황을 알고리즘을 이용해 미리 예측하여 자동으로 테스트를 수행할 수 있는 방안을 제시하였다.

논문에서 제안한 기법을 검증하기 위해 기존 기법과 제안하는 기법을 이용해 각각의 경우에 대한 테스트 케이스를 도출하였으며, 테스트를 수행해 결함의 수를 파악하였다. 도출된 결과를 이용해 각 모델에서의 결함 발견율을 계산한 결과를 통해 제안한 기법을 적용한 모델에서 결함 발견율이 7.08% 향상됨을 알 수 있었다. 이와 같은 결과를 통해 제안한 방법이 전문가의 경험에 의존하지 않고 자동 테스트 수행을 통해 예외상황에 대한 결함을 발견할 수 있다는 면에서 의미가 크다고 할 수 있다.

참 고 문 헌

[1] Chen Minsong; Qiu Xiaokang; Li Xuandong, Automatic test case generation for UML activity diagram, Proceedings of the 2006 international workshop on Automation of software test AST ’06.

[2] Debasish Kundu and Debasis Samanta, A Novel Approach to Generate Test Cases from UML Activity Diagrams, Journal of Object Technology, Vol 8, No. 3, May-Nume 2009.

[3] S. Lin and D. J. Costello Jr., Error control coding : fundamentals and applications, 2nd ed., Pearson Prentice Hall, pp.743-748, 2004,

[4] Huaizhong Li, Chiou Peng Lam, Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams, TestCom 2005, LNCS 3502, pp.69-80, 2005.

[5] Jerry Zeyu Gao, Jacob Taso Ye Wu, Testing and Quality Assurance for Component based Software, Chapter7, white-box testing methods for software components.

[6] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, Generating test cases from UML activity diagram based on gray-box method, In 11th Asia-Pacic Software Engineering Conference (APSEC04), pp. 284-291, 2004.

[7] C. Mingsong, Q. Xiaokang, and L. Xuandong, Automatic test case generation for UML activity diagrams, In 2006 international workshop on Automation of software test, pp. 2-8, 2006.

[8] Hyungchoul Kim, Generating Test cases from UML Activity Diagrams, Information and Communications University.

저 자 소 개

최 인 화

- 2005년 2월 : 서울여자대학교 컴퓨터학과(공학사)
- 2007년 2월 : 서울여자대학교 컴퓨터학과 (이학석사)
- 2008년 3월 ~ 현재 : 서울여자대학교 컴퓨터학과 이학박사과정
- 주관심분야 : 디지털방송, 소프트웨어 테스팅

저 자 소 개

조 민 주

- 2005년 2월 : 서울여자대학교 컴퓨터학과(공학사)
- 2008년 2월 : 서울여자대학교 컴퓨터학과 (이학석사)
- 2008년 3월 ~ 현재 : 서울여자대학교 컴퓨터학과 이학박사과정
- 주관심분야 : Convergency Computing, 디지털방송

백 종 호

- 1994년 2월 : 중앙대학교 전기공학과(공학사)
- 1997년 2월 : 중앙대학교 전기공학과(공학석사)
- 2007년 8월 : 중앙대학교 전자전기공학부(공학박사)
- 1997년 1월 ~ 2011년 8월 : 전자부품연구원 모바일단말연구센터 센터장
- 2011년 9월 ~ 현재 : 서울여자대학교 정보미디어대학 멀티미디어학과 교수
- 주관심분야 : 차세대 방송통신시스템, 차세대영상시스템, 소프트웨어 테스팅

황 준

- 1985년 8월 : 중앙대학교 컴퓨터공학과(공학사)
- 1987년 8월 : 중앙대학교 컴퓨터공학과(공학석사)
- 1991년 2월 : 중앙대학교 컴퓨터공학과(공학박사)
- 1992년 3월 ~ 현재 : 서울여자대학교 정보미디어대학과 멀티미디어학과 교수
- 주관심분야 : Convergency Computing, Digital Broadcasting