ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

임베디드 소프트웨어의 인터페이스 결함허용성 향상 기법☆

A Method for Improving Interface Fault Tolerance in the Embedded Software

최 인 화[1] 백 종 호[2*] 황 준[2]
In Hwa Choi Jong Ho Paik Jun Hwang

요 약

새롭게 개발되는 소프트웨어 컴포넌트와 재사용되는 하드웨어 컴포넌트를 결합할 때, 두 컴포넌트간 인터페이스 불일치현상이 발생할 수 있다. 인터페이스 불일치 현상은 다양한 형태의 결함을 야기할 수 있으며 인터페이스 결함허용성을 저하시키는 요인이 된다. 하지만 이러한 결함에 대한 테스트는 경험기반으로 수행되기 때문에 사람에 매우 의존적이라는 단점이 있다. 이에 본 논문에서는 경험기반 테스트를 구조적 방법으로 접근하여 임베디드 소프트웨어의 인터페이스 결함허용성을 향상시킬 수 있는 기법을 제안한다. 먼저, 기존에 정의된 인터페이스 결함유형에는 포함되지 않았던 경험기반으로 수행되었던 인터페이스 결함유형을 정의한다. 다음으로 새롭게 정의된 인터페이스 결함 유형을 고려한 테스트 수행 방법을 제시한다.

제안한 방법의 우수성을 입증하기위해 본 논문에서는 상용 방송 수신단말기를 대상으로 테스트 수행한 결과, 기존에 사용했던 방법보다 7.9%의 심각한 결함을 추가로 발견할 수 있었다. 또한, 제안한 방법은 소프트웨어 개발 주기상에서 초기에 테스트 경로를 생성하기 때문에 개발자들이 사전에 인터페이스 불일치 결함을 발견할 수 있어 보다 효과적인 테스트 계획 수립이 가능하다.

☞ 주제어 : 테스팅, 인터페이스 테스팅, 인터페이스 불일치 결함 테스트 모델, 결함허용성

ABSTRACT

Generally, there can be a interface discrepancy between the legacy hardware and the new software in combining new software component with reused hardware components in the embedded system. This kind of the interface discrepancy may cause various types of faults and also result in declining interface fault tolerance. In this paper we propose a method to improve interface fault tolerance. First of all, the new interface discrepancy fault type which has not been dealt with before is to be defined and next the testing method for generating test paths is proposed by considering the new defined interface discrepancy fault type in this paper.

Several tests show that the proposed method detects more fatal faults about 7.9% in comparison with the existing testing method for commercial broadcasting receiver. Since the proposed method can provide software developers with test paths to be available earlier on the software development cycle, in addition, software developers can regard on interface discrepancy fault in advance. Consequently, more efficient test planning can be established to improve the interface fault tolerance.

☞ keyword : Testing, Interface Testing, Interface Discrepancy Fault Test Model, Falut Tolernance

1. 서 론

최근 휴대폰이나 아이패드 또는 갤럭시 탭 등과같이 하나의 제품에 다양한 기능이 동시에 탑재되는 다양한 멀티컨버전스 제품들이 등장하면서 많은 사람들의 관심이 멀티컨버전스 제품에 집중되고 있으며, 제품에 대한 수요 또한 점점 빠르게 증가하고 있다. 이러한 변화와 함께 임베디드 제품 개발사에서는 수요를 충족시키기 위해 점점 더 다양한 제품을 보다 빠르게 생산할 수 있는 여러 방법들을 모색하고 있는데, 그 방법들 중 하나가 개발일정의 단축이다.

임베디드 시스템은 일반 패키지 소프트웨어와는 다른 특성들을 갖고 있다[1-3]. 따라서 임베디드 시스템을 개발할 때 일반적으로 특정목적을 수행할 수 있는 하드웨어 컴포넌트와 소프트웨어 컴포넌트를 동시에 개발하는

1 Department of Computer Science, Seoul Women's University, Seoul, 139-774, Korea
2 Department of Multimedia, Seoul Women's University, Seoul, 139-774, Korea
* Corresponding author (paikjh@swu.ac.kr)
[Received 3 September 2012, Reviewed 18 September 2012, Accepted 31 December 2012]
☆ 이 논문은 2012학년도 서울여자대학교 교내학술특별연구비의 지원을 받았음.
☆ 본 논문은 2012년도 한국인터넷정보학회 하계학술발표대회 우수논문의 확장버전임.

http://dx.doi.org/10.7472/jksii.2013.14.31

Co-design형태의 개발방식을 채택한다[4,5]. 하지만 이와 같은 방법으로는 제품개발일정을 단축하는데 많은 한계가 존재한다. 따라서 제품 개발사에서는 종전에 사용하였던 Co-design형태에 변화를 주어, 소프트웨어 컴포넌트는 새로 개발하고 하드웨어 컴포넌트는 타사에서 기 개발된 제품으로 재사용하여 두 컴포넌트를 결합하는 형태의 개발방식을 채택하기 시작하였다.

컴포넌트의 재사용은 제품 개발에 다양한 이점을 제공하는데 특히 본 논문에서 다루는 문제에 도움이 되는 개발일정 단축과 비용 절약측면에 큰 도움이 된다[6][7]. 그러나 타사에서 기 개발된 즉, 통합되어질 소프트웨어 컴포넌트와 상이한 요구사항을 기반으로 개발된 라이브러리 형식의 컴포넌트를 사용하기 때문에 소프트웨어 컴포넌트와 통합되는 인터페이스에 결함이 발생하였을 경우 제품의 신뢰성을 보장하기가 어려워진다는 문제점이 있다. 특히 인터페이스 결함허용성을 저하시키는 요인이 될 수 있다.

결함허용성이란 제품에 결함이 존재함에도 불구하고 사용자에게 문제없이 서비스를 제공하는 능력을 의미하는데, 하드웨어 컴포넌트를 재사용하는 임베디드 제품의 경우 인터페이스에 결함이 발생하면 두 컴포넌트 사이에 충돌이 발생하여 다양한 형태의 오류가 발생하기 때문에 사용자가 원하는 서비스를 제공할 수 없게 된다[8]. 그러나 현재 많은 임베디드 제품 개발사에서는 인터페이스 결함테스트가 거의 수행되지 않고 있다. 그 이유는 인터페이스 결함테스트가 경험을 기반으로 수행되는 테스트이기 때문이다.

구조적 테스트 설계기법은 요구사항을 분석하여 다이어그램을 도출하고 이를 통해 논리적으로 테스트를 수행할 수 있는 방법을 제공한다[9,10]. 반면, 경험기반 테스트는 테스트 수행자의 능력과 경험 그리고 직관에 매우 의존적이기 때문에 경험자가 없을 경우 테스트가 거의 수행될 수 없는데, 임베디드 제품 테스트 현실상 경험 있는 테스터는 매우 찾아보기가 어려운 실정이다. 만약 경험 있는 테스터가 존재하지 않는다고 하더라도 발생할 수 있는 인터페이스 결함에 대한 정의가 명확히 되어 있다면, 초보자라도 기술된 결함 정의를 참고로 하여 테스트를 수행할 수 있다. 하지만 결함에 대한 정의 또한 많이 부족한 상황이다.

이러한 문제들을 해결하기 위해 본 논문은 두 가지를 제안한다. 첫째, 인터페이스 결함유형의 정의이다. 정의되는 유형은 새로 개발되는 소프트웨어 컴포넌트와 타사에서 기 개발된 하드웨어 컴포넌트를 통합할 때 인터페이스에서 발생할 수 있는 결함유형이다. 둘째, 결함테스트의 구조적 테스트 방법을 제안한다. 앞에서도 언급했듯이 결함테스트는 경험기반테스트이기 때문에 논리적으로 테스트를 수행할 수 있는 방법이 정의되어 있지 않다. 따라서 경험에 의존하여 테스트를 수행하는데, 본 논문에서는 경험기반 테스트에 구조적 설계기법을 적용함으로서 논리적으로 테스트를 수행할 수 있는 기법을 제안한다. 제안기법은 실시간처리가 가능한 소형 임베디드 시스템의 소프트웨어를 대상으로 한다.

2. 관련연구

일반적으로 인터페이스 결함은 통합되는 두 소프트웨어 컴포넌트가 잘못된 값으로 상호작용할 때 발생한다. 그러나 임베디드 소프트웨어의 경우에는 잘못된 값이 하드웨어와의 통합을 통해서도 전달될 수 있으며, 소프트웨어 사이의 통합이라도 하드웨어에 탑재된 환경에서 테스트되기 때문에 하드웨어의 잘못된 상태에 간접 영향을 받을 수 있다. Delamaro와 Mathur는 소프트웨어 통합에서 일반적으로 발생할 수 있는 결함유형을 세 가지로 정리하였다. 그리고 서주영[11]은 소프트웨어와 하드웨어 통합을 고려하여 5가지의 유형을 추가로 정의하고 Delamaro와 Mathur의 연구와 통합하여 인터페이스 결함유형을 다음과 같이 정리하였다.

임베디드 시스템을 P, P의 테스트 케이스를 T라 할 때, P는 소프트웨어 컴포넌트 F와 G, 그리고 하드웨어 컴포넌트 H로 구성된다.

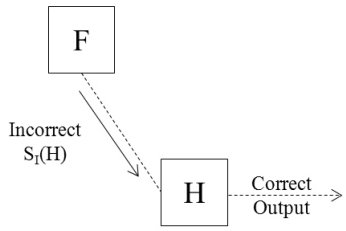
- F는 H를 호출하는 하드웨어 신호를 전송한다.
- H는 G로 하드웨어 신호를 전송한다.

SI(G)를 G로 전달되는 n개의 값이라 하고 So(G)를 G에 의해 전달되는 n개의 값이라 할 때,

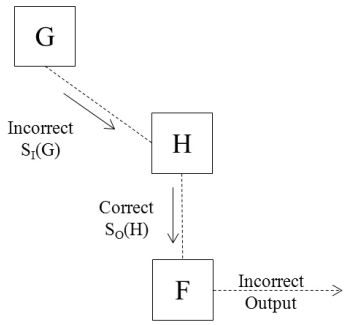
$S_I(G)$:함수 G의 호출에 사용되는 n개의 입력값은
→ 함수 호출에 사용되는 입력 파라미터
→ G에서 사용되는 전역변수에 의해 결정된다.

$S_o(G)$:함수 G의 의한 n개의 결과값은
→ 함수 호출에 사용된 결과 파라미터
→ G에서 사용되는 전역변수
→ G에서 리턴 되는 값에 의해 결정된다.

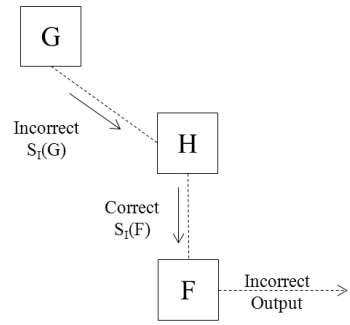
테스트케이스 t에 있는 P를 실행시켰을 때, F로부터 G



(그림 1) 결함유형 9
(Figure 1) Fault Type 9



(그림 2) 결함유형 10
(Figure 2) Fault Type 10



(그림 3) 결함유형 11
(Figure 3) Fault Type 11

의 호출에서 정의되는 인터페이스 결함은 다음과 같다.

- 유형 1: G의 도입부에서 $S_I(G)$는 예상치 못한 값을 갖고 이로 인해 G에서 리턴 값을 보내기 전에 잘못된 결과를 생산한다.
- 유형 2: G의 도입부에서 $S_I(G)$는 예상한 값을 갖는다. 그러나 G 내부의 실행 오류로 잘못된 $S_O(G)$값이 리턴되어 F에서 잘못된 결과를 생산한다.
- 유형 3: G의 도입부에서 $S_I(G)$는 예상치 못한 값을 갖게 되고 이는 G로부터 잘못된 $S_O(G)$값을 리턴하도록 유도한다. 그리고 그 영향으로 F에서 잘못된 결과를 생산한다.

테스트 케이스 t에 있는 P를 실행시켰을 때, F로부터 H로 보내는 신호로부터 발생할 수 있는 인터페이스 결함은 다음과 같다.

- 유형 4 : H의 도입부에서 $S_I(H)$는 예상치 못한 신호 값을 보낸다. 이 값은 H에서 리턴 값이 오기 전 잘못된 결과를(하드웨어 결함) 생산한다.
- 유형 5: H의 도입부에서 $S_I(H)$는 예상한 신호 값을 보내지만 H 내부의 실행 오류로 잘못된 SO(H) 값을 리턴 해 F에서 잘못된 결과를 생산한다.

테스트 케이스 t에 있는 P를 실행시켰을 때, H로부터 간접 영향을 받는 F로부터 G의 호출에서 정의되는 인터페이스 결함은 다음과 같다.

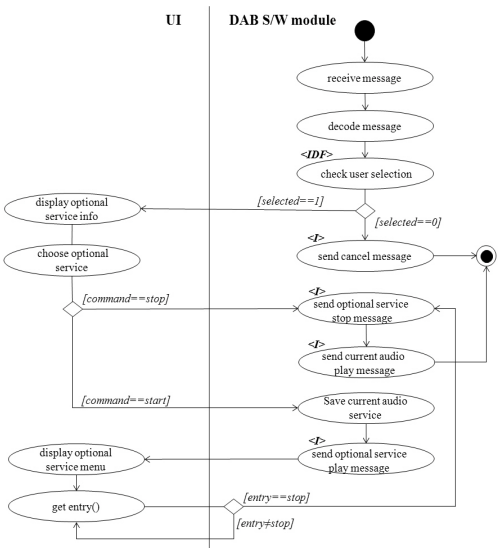
- 유형 6 : G의 도입부에서 $S_I(H)$는 예상한 값을 갖고 있어 올바른 결과를 생산할 수 있다. 그러나 H로부터 예상치 못한 값을 가진 $S_I(H)$를 받게 되고 이로 인해 G에서 H로 리턴 값을 보내기 전 G에서 잘못된 결과를 생산한다.
- 유형 7: G의 도입부에서 $S_I(G)$는 예상한 값을 갖고 있고, 이로 인해 올바른 $S_O(G)$를 F로 보낸다. 그러나 F는 H로부터 예상치 못한 값을 가진 $S_I(H)$를 받게 되고 이로 인해 잘못된 결과를 생산한다.
- 유형 8: G의 도입부에서 $S_I(H)$는 예상한 값을 갖고 있다. 그러나 H로부터 예상치 못한 값을 가진 $S_I(H)$를 받게 되고 이로 인해 예상치 못한 값의 $S_O(H)$를 F로 보낸다. F는 잘못된 결과를 생산한다.

3. 인터페이스 불일치 결함유형 정의

관련연구에서 기존에 정의된 인터페이스 결함유형 8가지를 살펴보았다. 정의된 유형은 소프트웨어와 소프트웨어, 소프트웨어와 하드웨어에서 발생할 수 있는 결함의 유형을 잘 정리하였다. 하지만 재사용되는 하드웨어 컴포넌트와 새로운 소프트웨어 컴포넌트가 통합될 때 발생할 수 있는 인터페이스 불일치 결함에 대해서는 고려되지 않고 있다. 본 논문은 이러한 결함 형태를 IDF (Interface Discrepancy Fault)라 하며 다음과 같이 3개의 유형으로 정리한다.

테스트 케이스 t에 있는 P를 실행시켰을 때, F로부터 H로 보내는 신호로부터 발생할 수 있는 인터페이스 결함은 다음과 같다.

- 유형 9 : H의 도입부에서 $S_I(H)$는 예상치 못한 값을 갖게 되고, 이로 인해 F가 의도하지 않은 값이 입력된다. 하지만 H는 정상적인 값으로 받아들여 다음 처리를 시도한다.
- 유형 10 : 유형 9의 결과로 H는 F가 의도하지 않은 기능을 수행하고 그 결과 $S_o(H)$에 의해 잘못된 값을 메모리나 레지스터에 저장한다. 이를 이용하는 컴



(그림 4) 속성을 부여한 액티비티 다이어그램
(Figure 4) Activity Diagram with Assigned Attributes

포넌트 G는 잘못된 결과를 생산한다.

- 유형 11 : 유형 9의 결과로 H는 F가 의도하지 않은 기능을 수행하고 그 결과 $S_I(G)$에 의해 컴포넌트 G에게 실행 명령을 전달한다. 컴포넌트 G는 잘못된 결과를 생산한다.

정의된 유형을 다음과 같이 도식화하여 표현할 수 있다.

4. 결함기반 테스트의 구조적 접근 방법

4.1 액티비티 다이어그램을 이용한 결정 포인트

제어흐름 테스트를 수행하기 위해서는 프로그램 설계 또는 기술적 설계가 테스트 베이시스로 사용되어야 한다. 본 논문에서는 액티비티 다이어그램을 테스트의 베이시스로 사용하며, 결함상황을 표현하기 위해 액티비티 다이어그램의 요소에 몇 가지의 속성을 부가한다. (그림 4)의 액티비티 중 <IDF>,<I>가 표기된 액티비티가 추가된 속성이다. <I> 속성은 하드웨어 인터페이스와 연결되는 액티비티에 부여되는 속성을 의미한며, <IDF> 속성은 결정 포인트 중 모든 <I> 속성의 액티비티와 연결되는 결정 포인트에 부여되는 속성을 의미한다. (표 1)에서는

(표 1) 인터페이스 목록
(Table 1) List of Interface

액티비티	인터페이스
send cancel message	I1
send optional service stop message	I2
send current audio play message	I3
send optional service play message	I4

(표 2) 결정 포인트와 연결되는 인터페이스 목록
(Table 2) List of Interface Connected to Decision Point

액티비티	결정 포인트	연결되는 인터페이스
Main Service	C1	I1, I2, I3, I4
Optional Service	C2	I2, I3, I4
Optional Service	C3	I2

<I> 속성을 갖고 있는 인터페이스 목록을 제시하고 (표 2)에서는 결정 포인트와 연결되는 인터페이스 목록을 나타낸다. (표 2)의 결정 포인트 C1은 인터페이스 속성을 갖고 있는 모든 액티비티와 연결 관계에 있고 C2는 I1을 제외한 3개의 인터페이스, 그리고 C3은 하나의 인터페이스 I2와 연결되었음을 볼 수 있다. 이들 중 모든 인터페이스와 관계를 맺고 있는 C1이 IDF 액티비티로 결정된다.

소프트웨어 컴포넌트의 결함 발생으로 다른 하드웨어 인터페이스가 호출될 때, 호출될 수 있는 하드웨어 인터페이스는 두 가지 유형이 존재한다. 하나는 액티비티 다이어그램에 표현된 하드웨어 인터페이스이고 다른 하나는 다이어그램에 표현되지 않은 하드웨어 인터페이스이다. 전자는 위에서 <I> 속성을 부여한 인터페이스이며 후자는 하드웨어 컴포넌트에서만 제공하는 인터페이스이다. 하드웨어 컴포넌트의 경우 전문가에 경험에 의한 테스트가 일반적이었으나 본 논문에서는 전문가의 경험과 지식이 없이도 체계적으로 테스트를 수행할 수 있는 방법을 제안한다. 이를 위해 하드웨어 컴포넌트에서만 제공하는 인터페이스 목록을 파악한다.

인터페이스 추가목록은 하드웨어 컴포넌트 개발사로부터 정보를 얻고, 소프트웨어 컴포넌트와 통합되는 인터페이스를 직접 확인하여 정확히 작성해야 한다. 표에서 추가된 하드웨어 인터페이스는 DI(Discrepancy Interface)라

고 구분하였다. 이 인터페이스는 다음 절 액티비티 다이어그램의 구조적 표현에서 인터페이스 결함 테스트 경로를 생성하는 하나의 노드가 된다.

4.2 액티비티 다이어그램의 구조적 표현

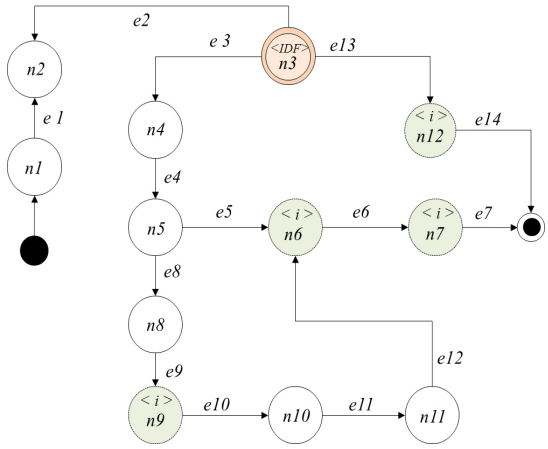
결함 테스트 경로를 포함한 모든 테스트 경로를 자동으로 생성하기 위해 위에서 작성한 액티비티 다이어그램을 기반으로 그래프를 생성한다. 그래프는 방향성 그래프로 다음과 같이 정의한다.

- 그래프 G는 “하나 이상의 노드 n의 집합 N과”과 “두개의 n을 연결하는 에지 e의 집합 E”로 구성
- n은 “고유 식별자 id 와 속성 a”로 구성
- a는 “인터페이스 노드를 나타내는 i, IDF 노드를 나타내는 IDF, 일반 노드를 나타내는 nidf 로 구성
- e는 “고유 식별자 id와 한 쌍의 n(tail, head), 그리고 한 쌍의 n을 성립시킬 수 있는 필요조건 c”로 구성

정의한 식을 이용해 액티비티 다이어그램을 그래프로 표현할 때 다음과 같은 사항을 고려한다.

- 각 액티비티는 하나의 노드로 표현한다.
- 노드에 고유의 식별자와 속성을 부여한다.
- 속성을 부여할 때, 액티비티가 하드웨어 인터페이스와 연결되는 경우 I 속성을, 하드웨어 인터페이스와의 연결 분기점이 되는 경우 IDF 속성을, 그 외의 경우 NIDF 속성을 부여한다.
- 액티비티 A에서 액티비티 B로의 연결정보 e를 표현할 때, 액티비티 A는 tail, B는 head로 표현한다.
- 만약 하나의 액티비티에서 조건에 의해 두 개 이상의 다른 액티비티로 분기된다면, 각각 연결되는 두 액티비티의 쌍으로 구성되는 에지의 조건으로 표현한다.

(그림 5)는 위에서 정의한 식과 고려사항을 기반으로 (그림 40에서 표현된 액티비티 다이어그램을 그래프로 표현한 결과로 (그림 4)의 12개의 액티비티는 그래프에서 정점 n1～n12로 표현되었다. 그리고 정점들 간의 연결정보는 e1～e14로 표현되었다. 다음의 N은 고유 식별자와 속성으로 구성된 12개의 노드 정보를 나타낸다.



(그림 5) 액티비티 다이어그램의 그래프 표현
(Figure 5) Grape Description of Activity Diagram

4.3 테스트 경로 자동생성 알고리즘

4.3.1 일반 테스트 경로 생성 알고리즘

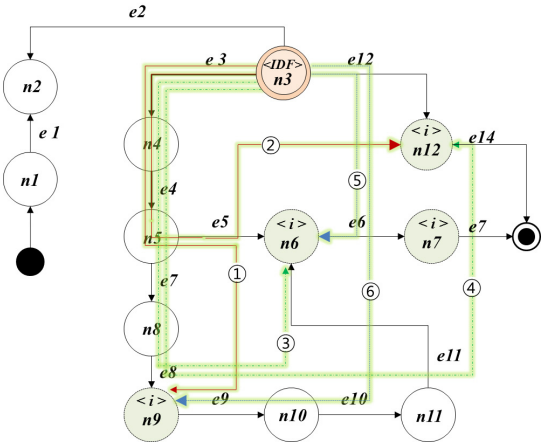
알고리즘을 적용할 소프트웨어 컴포넌트는 앞서 정의한 것처럼 하나 또는 그 이상의 소프트웨어 구성단위(module)로 조립된 개별적 동작이 가능한 최소 단위로서, 하드웨어 컴포넌트와 연결되는 기능을 포함하는 컴포넌트이기 때문에 순환구조가 존재하지 않는다. 따라서 일반 테스트 경로를 생성하는 알고리즘은 DFS(Depth First Search)알고리즘을 확장하였다. 이 알고리즘은 테스트 경로생성을 위한 주 알고리즘과 부 알고리즘으로 구성된다. 주 알고리즘에서는 경로를 하나 완성할 때까지 노드를 하나씩 방문하면서 스택에 경로를 저장하는 순서가 나타나며, 부 알고리즘에서는 하나의 경로가 완성되었을 때 IDF 노드의 합을 누적하며 생성된 경로를 따로 저장하는 흐름이 표현되었다. (표 3)은 테스트 경로생성 주 알고리즘의 세부내용의 일부를 나타낸다.

4.3.2 인터페이스 불일치 결함 테스트 경로 생성 알고리즘

IDF는 결함이기 때문에 액티비티 다이어그램에서 표현하지 않고 두 번째 단계인 그래프를 이용해 발생경로를 예측하여 테스트 경로를 생성한다. (그림 6)은 그래프에서 발생할 수 있는 IDF를 나타내었다.

(표 3) 테스트 경로생성 주 알고리즘
(Table 3) Main Algorithm of Test Path Generation

[테스트 경로생성 주 알고리즘]
1. 특정노드를 시작점으로 선택하여 탐색을 시작 한다. 이 노드를 node1이라 칭한다.
2. 선택된 node1에 방문표시를 한다.
3. node1을 tail로 하는 연결정보 e를 검색한다.
4.1 e가 없을 경우 6번을 수행한다.
4.2 e가 있을 경우, e에 속한 head노드가 end노드인지 확인한다.
5.1 head노드가 end노드이면 부 알고리즘을 수행한다.
(...중략)
6. 최종적으로 삽입된 노드를 스택으로부터 꺼내 node1이라 칭한다.
7. MIN에 스택 포인트를 저장한다.
8. 3번부터 다시 수행한다.



(그림 6) IDF 테스트 경로의 그래프 표현
(Figure 6) Grape Description of IDF Test Path

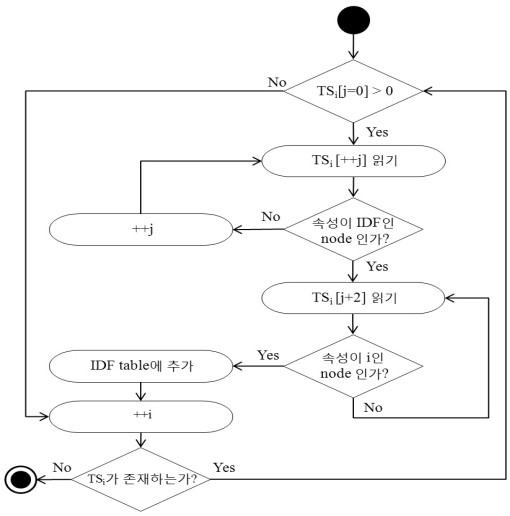
그림을 보면 전체적으로 6개의 IDF 경로가 존재함을 알 수 있다. 경로 ⑥에서 발생하는 결함을 그래프 상에서 살펴보면, 소프트웨어 컴포넌트는 n1 → n2 → n3 → n12 의 경로를 수행한다. 그런데 결함발생으로 인해 인터페이스 n12로 가야할 경로가 조건이 다른 인터페이스인 n9로 바뀌어 들어가는 경우이다. 경로 ⑤도 경로 ⑥과 마찬가지로 n1 → n2 → n3 까지는 바르게 수행되고 인터페이스 n12가 아닌 다른 조건의 인터페이스 n6이 수행되는 경우를 나타낸다. ⑤ 와 ⑥의 결과 하드웨어 컴포넌트에서 의도하지 않는 실행을 하게 되고 소프트웨어 컴포넌트가 그 영향을 받을 때, 소프트웨어의 잘 설계된 오류처리로 큰 결함의 발생을 막을 수 있다. 하지만 n6이나 n9 사이에 수행하지 않은 n4, n5, n8이 연결된 필수기능일 경우는 또 다른 문제가 발생할 수 있다. IDF 경로 ①은 n1 → n2 → n3 → n4 → n5 → n6 의 경로를 수행해야 하는데 결함 발생으로 n1 → n2 → n3 → n4 → n5 → n9 로 잘못 수행된 경우를 나타낸다. 이 경우는 처리 경로가 원래의 경로와 유사하기 때문에 하드웨어로부터 파생되는 예상되지 못한 결과에 대해 소프트웨어 컴포넌트가 오류처리를 잘 해낼 경우 n8의 기능과의 관련성에 의해 간단하게 처리될 수도 있는 경우에 해당된다.

경로 ②는 n1 → n2 → n3 → n4 → n5 → n6 의 경로를 수행해야 하는데 결함 발생으로 n1 → n2 → n3 → n4 → n5 → n12 로 잘못 수행된 경우를 나타낸다. 이 경우는 소프트웨어 컴포넌트에서 이미 n12와는 관련 없는 n4와 n5를 실행했기 때문에 n4와 n5의 기능에 따라 발생하는 오류의 파생범위가 달라질 수 있다.

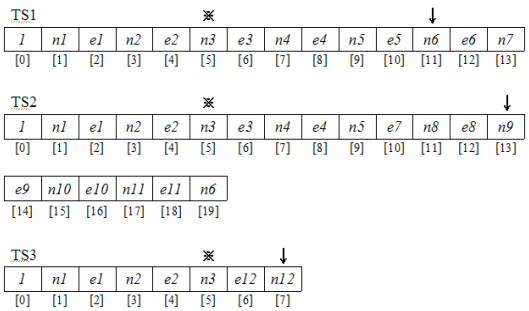
경로 ③은 n1 → n2 → n3 → n4 → n5 → n8 → n9 의 경로를 수행해야 하는데 결함 발생으로 n1 → n2 → n3 → n4 → n5 → n8 → n6 으로 잘못 수행된 경우를 나타낸다. 이 경우는 ①, ②, ⑤, ⑥의 경우와는 달리 하드웨어 파생결과에 대해 소프트웨어 컴포넌트가 오류처리를 잘 해낼 경우 수행경로가 동일하기 때문에 가볍게 처리될 수 있다. 경로 ④는 n1 → n2 → n3 → n4 → n5 → n8 → n9 의 경로를 수행해야 하는데 결함 발생으로 n1 → n2 → n3 → n4 → n5 → n8 → n12 으로 잘못 수행된 경우를 나타낸다. 이 경우 위의 경로 ③과 마찬가지 경우에 해당된다.

도출된 IDF 경로들을 살펴보면 모두 인터페이스 노드와 관련성이 있으며, 특정 조건에 의해 인터페이스 노드를 결정짓는 노드로부터 파생되는 것을 알 수 있다.

(그림 7)의 알고리즘을 테스트 경로에 적용하여 IDF 테이블을 생성하면 1개의 IDF 노드와 3개의 인터페이스 노드를 찾을 수 있다. 생성된 테이블은 IDF 테스트 경로를 생성할 때 각각의 테스트 경로마다 적용되는 중요한 자료이다. (그림 8)에서는 테이블을 생성하기 위해 알고리즘을 수행하는 과정에서 실제 데이터의 변화 흐름이 나타난다. (그림 8)의 TS1에서는 처음 만나는 IDF 노드가 n3이며 인터페이스 노드는 n6이다. TS2는 IDF 노드 n3에 인터페이스 노드는 n9, 그리고 TS3는 IDF 노드 n3에 인터페이스 노드 n12를 찾을 수 있었다. 그림에서 IDF노드는 ※ 표시를 하였고 인터페이스 노드에는 ↓ 표시를 하였다.



(그림 7) IDF 테이블 생성 알고리즘
(Figure 7) Generation Algorithm of IDF Table

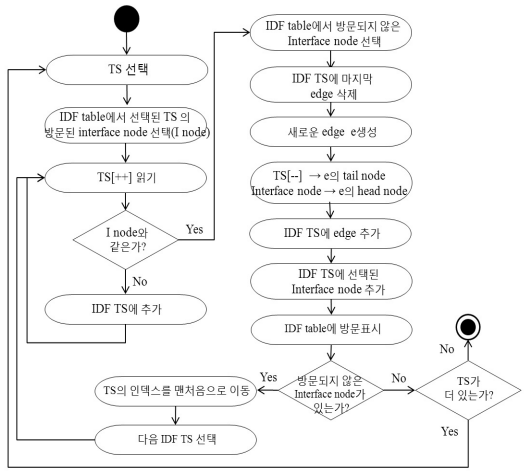


(그림 8) IDF 테이블 생성 알고리즘에서의 데이터 흐름
(Figure 8) Data Flow on Generation Algorithm of IDF Table

생성된 IDF 테이블을 기반으로 IDF 테스트 경로를 생성한다. 테스트 경로는 (그림 9)에 나타난 알고리즘을 이용해 생성된다.

5. 테스트 수행결과

앞서 제안한 테스트 기법을 상용제품의 전 기능에 대해 테스트를 수행하였으며, 제안한 테스트 모델의 비교분석을 위해 제품 개발업체에서 기존에 사용하던 테스트 모델과 병행하여 실험을 진행하였다. 테스트 수행결과는 테스트 경로의 수와 테스트 수행 후 발견된 결함의 수,



(그림 9) IDF 테스트 경로 생성 알고리즘
(Figure 9) Generation Algorithm of IDF Test Path

(표 4) 테스트 수행결과 비교표
(Table 4) Comparison of Testing Results

	이전모델		제안 모델	
	TP	fault	TP	fault
connection	12	0	12	0
configuration	12	3	13	5
frequency setting	16	4	22	5
scanning	32	4	33	8
information	14	5	16	7
seek	30	3	47	10
diag info	9	0	9	0
reset	4	0	6	0
dab to dab	34	5	40	5
announcement	22	3	34	15
dab to fm	35	7	38	8
reconfiguration	50	12	68	22
DLS	14	5	16	5
alarm	4	0	4	0
zber	6	0	6	0
favorite	35	7	48	15
preset	12	3	16	5
FM	10	0	15	6
DRC1	6	0	10	0
channel database	6	1	10	1
service follow	12	4	23	12
총 계	375	66	486	129

그리고 결함의 특성을 평가기준으로 삼았다.

(표 4)에서 두 모델에 대한 테스트 수행결과를 볼 수 있다. (표 4)에서 사용된 항목 중 기존모델은 개발업체에

서 기존에 사용하던 테스트 모델을 의미하며 제안 모델은 본 논문에서 제안한 방법을 사용한 모델을 의미한다.

6. 결 론

최근 멀티 컨버전스 기능이 탑재된 임베디드 제품의 수요가 급증하면서 개발일정을 단축시키기 위해 기 개발된 하드웨어 컴포넌트를 재사용하는 현상이 늘고 있다. 컴포넌트의 재사용은 재사용되는 컴포넌트가 다른 컴포넌트와 통합될 때 정상적인 기능수행이 검증된다면 시간과 비용을 절약할 수 있는 매우 좋은 방법이다. 그러나 현재 임베디드 제품의 경우 통합과정에서 인터페이스 불일치 결함이 발생해 제품의 결함허용성을 저하시키는 요인이 되고 있다. 하지만 이와 같은 결함은 경험을 기반으로 발견하기 때문에 사람에 매우 의존적이라는 문제가 있었다. 본 논문에서는 이와 같은 문제를 해결하기 위해, 경험기반으로 수행되던 방법을 구조적 방법으로 접근하여 인터페이스 불일치 결함이 발생할 때 제품의 결함허용성을 향상시킬 수 있는 방안을 제시하였다. 먼저, 인터페이스 불일치 결함 유형을 정의하였다. 논문에서 정의된 유형은 기 정의된 결함 유형으로는 표현하기 어려웠던 불일치 결함에 대한 명확한 정의이기 때문에 테스트 수행자에게 통찰력을 제공하여 간접적인 시스템 품질 향상에 중요한 요소로 작용할 수 있다. 다음으로, 인터페이스 결함유형에 대한 테스트 방법으로 구조적 테스트 설계기법을 적용하여 경험기반 테스트를 논리적으로 수행할 수 있도록 제안한 방법이다. 기존의 인터페이스 결함 테스트는 경험자의 경험과 직관에 의존적으로 수행되었지만 본 논문의 제안방법은 결함테스트를 구조적 방법을 사용해 수행하기 때문에 테스터의 자질과 상관없이 자동화에 의해 논리적으로 테스트를 수행할 수 있는 방법이다.

논문에서 제안한 테스트 기법의 우수성을 검증하기 위해 상용 방송 수신 제품을 대상으로 테스트를 직접 수행하고 결과를 타제품과 비교하여 분석한 결과 다음과 같은 결과를 얻을 수 있었다. 첫째, 제안한 기법을 사용함으로써 경험 있는 전문테스터가 없이도 인터페이스 불일치에 대한 테스트를 수행할 수 있었다. 둘째, 테스트 수행결과 타 제품보다 59%의 결함을 더 발견하였으며, 발견된 결함 중 7.9%는 화면이나 기능이 정지해버리는 심각한 실행문제를 야기하는 결함을 확인할 수 있었다.

참 고 문 헌(Reference)

[1] J. hugues, B. Zalila, L. Pautet, "Parid prototyping of the final embedded system using the ocarina AADL tool suite", ACM Transactions on Embedded Computing Systems(TECS), Vol. 7, Issue 4, Article No.42, 2008

[2] P. Liggesmeyer, "Trends in embedded software engineering", IEEE Software, Vol. 26, pp.19-25, 2009

[3] T. Wei-Tek, T. Lian, Z. Feng, and P. Ray, "Rapid embedded system testing using verification patterns", IEEE Software, pp.68-75, 2005

[4] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.p. Joo, "PeaCE: A Hardware-software codesign environment for multimedia embedded systems", ACM Transactions on Design Automation of Electronic Systems, Vol. 12, No.3, Article 24, 2007

[5] A. A. Jerraya and W. Wolf, "Hardware/software interface codesign for embedded systems", IEEE Computer, pp.75-84, 2005

[6] J. Li, H.C. Zhang, Z. Lin, "Asymmetric negotiation based collaborative product design for component reuse in disparate products", Computers&Industrial Engineering, Vol. 57, pp 80-90, 2009

[7] A. McVeigh, J. Kramer, J. Magee, "Using resemblance to support component reuse and evolution", SAVCBS '06 Proceedings of the 2006

[8] R. Isermann, "Fault-diagnosis systems: an introduction from fault detection to fault tolerance", Springer, 2006

[9] G. M. Kapfhammer, M.L. Soffa and D. Mosse, "Testing in resource constrained execution environment", In proc. Of the Int'l Cof. On Automated Software Engineering(ASE), pp.418-422, 2005

[10] K. G. Larsen, M. Mikucionis and B. Nielsen and A. Skou, "Testing real-time embedded software using UPPAAL-TRON", In Proc. Of the Int'l Conf. on Embedded Software(EMSOFT), pp.299-306, 2005

[11] J. Y. Seo, Embedded Software Interface Test Based on the Status of System, Ph.D Thesis, Ewha Women's University, 2009

◐ 저 자 소 개 ◑

최 인 화
2005년 서울여자대학교 컴퓨터학과(공학사)
2007년 서울여자대학교 대학원 컴퓨터학과(이학석사)
2012년 서울여자대학교 대학원 컴퓨터학과(이학박사)
관심분야 : 차세대 방송통신 시스템, 소프트웨어 테스팅, SSPL
E-mail : urangi@swu.ac.kr

백 종 호
1994년 중앙대학교 전기공학과(공학사)
1997년 중앙대학교 대학원 전기공학과(공학석사)
2007년 중앙대학교 대학원 전자전기공학부(공학박사)
1997년~2011년 전자부품연구원 모바일단말연구센터 센터장
2011년~현재 서울여자대학 멀티미디어학과 교수
관심분야 : 차세대 방송통신 시스템, 차세대 영상시스템, 소프트웨어 테스팅
E-mail : paikjh@swu.ac.kr

황 준
1985년 중앙대학교 컴퓨터공학과(공학사)
1987년 중앙대학교 대학원 컴퓨터공학과(공학석사)
1991년 중앙대학교 대학원 컴퓨터공학과(공학박사)
1992년~현재 서울여자대학 멀티미디어학과 교수
관심분야 : Convergence Computing, Digital Broadcasting.
E-mail : hjun@swu.ac.kr