객체 지향 프로그램을 위한 제약조건을 갖는 객체 상태 행위 테스팅 기법

이창영[0] 이인혁 구연설
충북대학교 전자계산학과
{chylee,leerg}@selab.chungbuk.ac.kr, yskoo@cbucc.chungbuk.ac.kr

Object State Behavior Testing Technique with Constraints for Object Oriented Program

Chang-Young Lee[0] Lirenge Yeon-Seol Koo
Dept. of Computer Science, Chungbuk National University

요 약

객체 지향 소프트웨어에 있어서, 테스팅의 중요성은 전통적인 소프트웨어의 개발에 있어서의 중요성과 크게 다르지 않다. 테스팅이 부적절하게 수행된 경우 프로그램의 버그를 성공적으로 검출할 수 없으며, 소프트웨어 품질을 보장할 수 없다. 즉, 성공적인 소프트웨어를 위한 문제가 바로 테스팅이라 할 수 있다. 따라서 개발된 객체 지향 소프트웨어의 신뢰성을 향상시킬 수 있는 객체 지향 테스팅 방법이 필요하다.

이 논문에서는 전통적인 소프트웨어 테스팅 기법과 객체 지향 특성을 결합하여 객체 지향 소프트웨어 테스팅을 위한 단위 설정을 위해 실시간 시스템에서 하나의 행위를 정의할 수 있는 객체 메소드의 결합에 대한 이벤트 그래프(Eent Gaph)와 제약적 메소드 시퀀스(Mthod Sequence with Cnstraints)를 정의하고, 제약사항을 포함하는 이벤트 그래프와 메소드 시퀀스(sequence)를 엘리베이터 시스템에 적용하여 객체 지향 실시간 시스템에 대한 객체 상태 행위 테스팅(object state behavior testing)을 통해 인트라 클래스 테스팅 및 인터 클래스 테스팅 기법을 제안한다.

1. 서론

객체 지향 패러다임이 소프트웨어 산업에서 급속하게 채택되어 사용되어 왔지만 객체 지향 프로그램의 테스팅과 유지보수에 있어서 문제점를 야기시킨다. 비주얼 프로그램에 대한 관심이 집중되면서 이에 대한 연구가 활발히 진행중이다.

전통적인 테스팅 기법을 적용할만한 객체 지향 소프트웨어는 기능적 분해로 설명이 가능하고, 폭포수(waterfall) 라이프 싸이클로 개발되고, 세가지 단계의 테스팅 전략으로 구성된다. 즉, 객체 지향 언어의 객체 내에서 개개의 메소드는 명령적이다. 이는 모든 객체 지향 언어의 메시지 호출이 있으면, 메시지에 대한 메소드를 수행한 후 호출을 한 객체에 제어를 복귀함을 의미한다[1][6].

이 논문에서는 객체 지향 소프트웨어의 테스트 케이스 생성에 대한 중요성을 증가시키고, 전통적 소프트웨어 개발 및 테스팅의 이론적 구조 및 기법을 수정 또는 대치하여 객체 지향 소프트웨어에 적용한다. 2 절에서는 객체 지향 테스팅 기법의 관련 연구로서 객체 지향 테스팅 전략과 객체 지향 테스팅 기법 및 문제점을 기술하고, 3 절에서는 제안된 객체 상태 행위 테스팅 기법을 위한 이벤트 그래프(EG)와 제약적 메소드 시퀀스(MSC)를 정의하고 이를 엘리베이터 모델에 적용하여 모달 공식을 통해 검증하며, 4 절에서는 결론 및 향후 연구 내용을 기술한다.

2. 객체 지향 테스팅

2.1 객체 지향 테스팅 전략

테스팅은 실행하기 전에 미리 버그를 줄이고 버그 발견이 유용한 테스팅을 수행하도록 하는 버그 예방의 목적과 버그를 발견하여 디버깅하는 역할을 수행한다.

행위적 특성을 객체 지향 소프트웨어 시스템 개발에 확장할 경우 테스팅 방법은 다음과 같다[2][7].

A. 인트라 메소드(Intra Method)
   클래스의 메소드 내에서 definition-use path를 테스팅
B. 인터 메소드(Inter Method)
   하나의 퍼블릭(public) 메소드에 의해 호출되는 클래스의 메소드를 통해 클래스 변수의 정의-사용 패스를 테스팅
C. 인트라 클래스(Intra Class)
   하나의 클래스의 퍼블릭 메소드 호출에 교차하는 클래스 변수의 정의-사용 패스를 테스팅
D. 인터 클래스(Inter Class)
   다수의 클래스의 메소드 호출에 교차하는 클래스 변수의 정의-사용 패스를 테스팅

A와 B에 대한 문제는 기존의 구조적 테스팅 기법을 사용하여 해결 가능하고, C와 D를 해결하기 위해서는 객체 지향 테스팅 테크닉이 필요하다[3].

2.2 객체 지향 테스팅 기법 및 문제점

객체 지향 테스팅은 객체 지향 패러다임에 따른 상속성, 다형성(동적 바인딩), 캡슐화로 기인하는 문제를 효과적으로 테스팅하는 것이다.

상속 계층에서 상위 레벨의 행위는 하위 레벨에서의 올바른 행위를 보장하지 않는다. 동적 바인딩의 다형성은 수행 가능한 테스트케이스의 수를 증가시키므로 전통적인 테스팅의 단점인 패스 명시를 위한 원시 코드의 정적 분석은 객체 지향 패러다임에 도움이 되지 않는다. 효과적인 범위(scope)를 제안할 때, 캡슐화는 실행 상태의 제어성과 관측성에 있어 문제를 야기한다. 그러므로 Visual Basic/C++, Eiffel, Smalltalk 등의 객체 지향 언어로 작성된 프로그램의 테스팅이 필요하다[1].

이러한 문제에 대해 Binder는 FREE(Flattened REgular Expression) 테스팅 방법론을 통해 객체 지향 실행의 유닛, 통합, 시스템 테스팅을 위한 응집력있는 프레임웍(framework)을 지원하는 상태-기반 접근 방법을 연구 발표하였으며[7], Jogenson & Erickson은 시스템 응용에 있어서 MM-path와 Automic System Function 컴포넌트 특성에 기준한 통합 테스팅에 대한 전략을 소개하였는데, 이는 전통적인 소프트웨어 개발 기법의 특성을 객체 지향 테스팅으로 적용한 것이다[1].

Kung, Gao, Hsa, Toyoshima는 객체 지향 테스팅의 문제점을 기술하였으며[5], Murphy, Townsend, Wong는 클래스 & 클래스 클러스터 통합에 중점을 둔 수년간의 실제 테스트 경험을 발표하였다[2]. 또한, Poston은 객체 지향 명세화로 부터 어떻게 클래스 테스트 케이스를 자동으로 생성하는지를 데모하였고, McGregor & Korson은 시스템 측면의 통합 요구와 상호적·점진적 개발에 대한 관점을 소개하였다[6].

이러한 연구를 통한 요점은 소프트웨어 테스팅은 기본적으로 행위(what it does)와 관계하고 구조(what it is)와는 관계하지 않음을 알 수 있다. 또한 고객은 행위적 입장에서 소프트웨어를 이해하기 때문에 객체 지향 테스팅 구조는 행위적이라 할 수 있다.

3. 객체 상태 행위 테스팅

객체 메소드는 명령적인 언어에서 사용되며, 하나의 응집력있는 기능을 수행한다. 이는 메소드 시퀀스와 일치하며, 전통적인 테스팅 기법이 메소드 시퀀스와 시퀀스 상속에 대한 테스팅은 인터 클래스 레벨이고, 객체의 메소드 실행은 이벤트에 의해 발생하며, 제약조건을 갖는 데이터 멤버에 의해 결국 메시지는 종료된다.

객체 지향 소프트웨어의 객체 상태 행위 테스팅을 위해 EG(event graph)와 MSC(method sequence with Constraints)를 정의한다. 이벤트 그래프는 객체 지향 프로그램의 제어 흐름을 정의하는 것이고, 제약적 메소드 시퀀스는 객체 지향 프로그램의 데이터의 흐름을 제어하기 위한 것이다.

정의 1

EG는 객체의 추상화된 제어 흐름을 표현한다. EG의 노드는 객체 행위를 나타내는 제어 흐름을 표현한다.

EG = (N, E, i, t)
N은 EG에서 노드(node)의 집합을 표현,
E는 EG에서 에지(edge)의 집합을 표현,
i는 initial node를 표현,
t는 termination node를 표현.

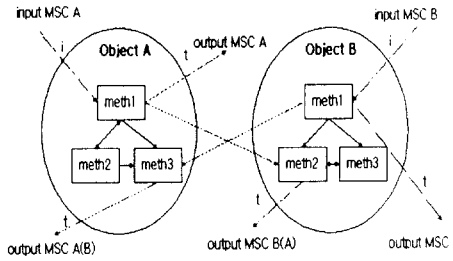

그림 1 이벤트 그래프

그림에서 Circle은 object를 나타내고, rectangle은 method를 나타내고, solid arrow는 edge를 dashed arrow는 객체사이의 method sequence를 나타낸다.

이벤트 그래프를 이용한 엘리베이터 시스템의 행위에 대한 객체 상태는 그림 2와 같다.

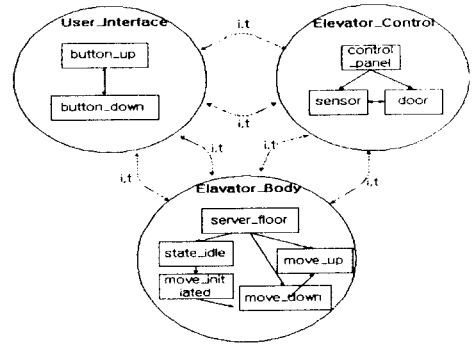

그림 2 엘리베이터 모델의 이벤트 그래프

엘리베이터 객체에 대해 시스템의 상태를 나타내는 행위를 메소드 시퀀스로서 나타낼 수 있다.

정의 2

MSC는 객체 지향 프로그램 테스팅을 위한 제약적 데이터 흐름을 나타내기 위한 것이다.

[States]
- 대기상태(state_idle)
- 버튼이 눌러진 층으로 엘리베이터 초기화 (move_initiated)
- 위층으로 올라가는 동작(move_up)
- 아래층으로 내려가는 동작(move_down)
- 문을 열고, 사람이 타거나 내리고, 문을 닫는 동작(serve_flooor)

[Constraints]
- floor button; [M1]
  : 현재 층 버튼이 눌러진 조건
- director button; [M2]
  : 목적지 버튼이 눌러진 조건
- Serve floor; [M3]
  : 현재 층에서 문을 열고 닫음
- X < N; [C1]
  : 현재 층이 목적지 층보다 낮은 조건

- X = N; [C2]
  : 현재 층이 목적지 층과 같은 조건
- X > N; [C3]
  : 현재 층이 목적지 층보다 높은 조건

객체 상태의 유효한 제약조건은 다음과 같다.

STATE	Valid Constraints
stay_idle	┐move_initiated
move_initiated	┐move_idle
move_up	move_initiated∧(┐serve_floor) ∨ ┐move_down)
move_down	move_initiated∧(┐serve_floor) ∨ ┐move_up)
serve_floor	move_initiated∧(┐move_up) ∨ ┐move_down)

여기서 ┐, ∧, ∨는 모달 로직(modal logic)을 따른다. f, g가 모달 공식이면, ┐f, ┐g, f∧g, f∨g도 모달 공식을 따른다.

시스템의 행위는 조건 값의 변화에 의해 결정되고 제어된다. 상태에 대한 조건 값들의 변화가 이벤트이며, 이벤트는 발생 시간 및 장소에서 검출 가능하다.

다음은 제약적 메소드 시퀀스이다.

- MSC @T(cond1)
  cond1의 값이 Falst에서 True로 변경될 때
- MSC @T(cond1) WHEN [cond2]
  cond2가 만족되고, cond1의 값이 F에서 T로 변경될 때
- MSC @T(cond1) ∧ MSC @T(cond2)
  cond1의 값이 T로 변경되고 cond2의 값이 T로 변경될 때

객체 상태 행위를 이와 같이 모달 공식을 사용하여 규격화함으로서 시스템 상태의 유효한 제약조건을 자동화하여 체크할 수가 있다. 이는 객체 지향 패러다임으로 구현된 프로그램의 테스팅을 위해 중요한 사항인 테스트 케이스를 생성하는 문제점에 있어서, 테스트 비용(test cost)을 고려할 경우에 테스트 케이스의 수를 현저하게 감소시킬 수 있다.

엘리베이터의 객체 상태 행위 테스팅을 위해 제약적 메소드 시퀀스를 고려하면 다음과 같다.

- state_idle
  : MSC @T(C2) WHEN [M1] ∨ MSC @T(3)
- move_initiated
  : MSC @T(M1∨M2)
- serve_floor
  : MSC @T(C2) WHEN [M1∨M2]
- move_up
  : MSC @T(C3) WHEN [M2∧M3]
  ∨ MSC @T(C3) WHEN [M2∨M3]
- move_down
  : MSC @T(C1) WHEN [M1∧M2∧M3]
  ∨ MSC @T(C1) WHEN [M1∨M3]

4. 결론

객체 지향 프로그램 개발에 있어서 가장 중요한 문제점은 테스팅 비용을 줄이는 것이다. 테스팅 비용은 프로그램 개발 전체 비용 중 상당한 비중을 차지한다고 볼 때, 객체 지향 프로그램으로 코딩된 소프트웨어의 테스트 케이스를 생성하기 위한 테크닉은 중요하다.

이 논문에서는 전통적인 테스팅 테크닉의 제어 흐름과 데이터 흐름을 고려한 테스팅 기법을 객체 지향 프로그램에 적용하여 객체 지향 행위 테스팅을 위한 이벤트 그래프와 제약적 메소드 시퀀스를 정의하였고, 이를 규격화를 이용하여 검증하였다.

향후 연구 방향은 객체 상태 행위 테스팅 기법을 적용하여 복잡한 시스템에 대한 체계적인 학습을 토대로 테스트 케이스의 생성을 자동화하고자 한다.

5. 참고 문헌

[1] Paul C. Jorgensen and Carl Erickson, "Object-Oriented Integration Testing", ACM, vol. 37, no. 9, September 1994, pp. 30~38

[2] Gail C. Murphy, Paul Townsend, Pok Wong, "Experiences with Cluster and Class Testing", comm. of ACM, vol.37, no.9, Sep.1994, pp.59~77

[3] M. D. Smith and D. J. Robson, "A Framework for Testing Object-oriented Programs", Journal of OOP, Vol. 5, no. 3, June 1992, pp. 45~53

[4] C. D. Turner and D. J. Robson, "The State-based Testing of Object-Oriented Program",Con. on Soft. Main., L .Alamitos, IEEE Computer Society press, 1993, pp.302~310

[5] David Kung, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, Chris Chen, Young-si Kim, and Young-Kee Song, "Developing an Object-Oriented Software Testing and Maintenance Environ-ment",ACM, vol. 38, no. 10, Oct. 1995, pp. 75~86

[6] John D. McGregor and Timothy D. Korson, "Integrated Object-oriented Testing and Devel-opment Processes", ACM, vol. 37, no. 9, September 194, pp. 59~77

[7] Robert V. Binder, "Object-Oriented Testing : Myth and Reality", ObjectMagazine, May 1995

[8] Marry Jean Harrold, John D. McGregor, and Kevin J. Fitzpatrick, "Incremental Testing of Object-oriented Class Structures", Proceedings, 14th IC on Software Engineering, LosAlamitos, IEEE Computer Society Press, May, 1992, pp. 68~80

[9] Tetsuro Katayama, Eisuke Itoh, Zengo Furukawa, "Test-case Generation for Concurrent Programs with the Testing Criteria Using Interaction Sequences", IEEE, 1999, pp 590-597

[10] 이창영, "도메인 분석을 통한 이벤트 의미 정형화", 석사학위논문, 충북대학교, 96