제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

소프트웨어 시험성 강화를 위한 테스트 오러클 생성 지원 환경

신동익, 전태웅
고려대학교 전산학과
e-mail : {eastwing, jeon}@selab.korea.ac.kr

Test Oracle Generation Support Environments for the High Testability of Software

Dong-ik Shin, Taewoong Jeon
Dept. of Computer Science, Korea University

요 약

소프트웨어 시험은 소프트웨어의 신뢰성을 직접적으로 향상시킬 수 있는 방법 중의 하나이지만 일반적으로 상당히 많은 비용이 드는 개발 과정이다. 따라서 경제적인 소프트웨어 개발을 위하여 소프트웨어 시험성을 강화시킬 수 있는 메커니즘들이 요구된다.

본 논문은 소프트웨어 시험성 강화 메커니즘들 중의 하나인 테스트 오러클의 생성을 지원하는 시험 환경의 구축 방법을 제안한다. 본 논문에서 제안하는 테스트 오러클 생성 지원 환경의 목적은 Statechart로 기술된 시험 대상 소프트웨어의 행위 모델로부터 실행 가능한 테스트 오러클의 생성을 지원하는 것이다.

1. 서론

소프트웨어의 응용 범위가 점점 다양해지고 특히, 중요한 응용 분야로 확대되어짐에 따라 소프트웨어 신뢰성이 더욱 중요해지고 있다. 이는 소프트웨어 신뢰성을 향상시킬 수 있는 방법들의 필요성이 더욱 커지고 있다는 것을 의미이다.

소프트웨어 신뢰성을 직접적으로 향상시킬 수 있는 방법들 중의 하나가 소프트웨어 시험이다. 시험을 통해 소프트웨어 내에 존재하는 오류들을 찾아냄으로써 소프트웨어 신뢰성은 향상될 수 있다.

소프트웨어 시험은 SUT(Software Under Test:시험 대상 소프트웨어)에 입력값들을 제공하고 입력값들에 대한 결과값들을 평가하는 과정이다[1]. 따라서 소프트웨어 시험을 위해서는 SUT에 대한 입력들 뿐만 아니라 이들 입력들에 대한 기대 결과들이 요구된다. 다시 말해, 소프트웨어 시험을 위해서는 SUT의 테스트 입력과 출력 결과들을 도출하는데 필요한 정보가 어떤 형태로든 명세되어 있어야 한다. 만약, SUT에 대한 명세가 정형화되어 있다면 SUT에 대한 테스트 입력 및 기대 결과의 도출 또는 도출의 자동화에도 많은 도움이 될 것이다.

소프트웨어 시험은 일반적으로 상당히 많은 비용이 드는 소프트웨어 개발 과정이다. 따라서 경제적인 소프트웨어 개발을 위해서는 소프트웨어 시험에 들어가는 비용을 줄일 수 있는 방법, 다시 말해 소프트웨어 시험성을 향상시킬 수 있는 방법들이 필요하다.

소프트웨어 시험성은 소프트웨어가 시험 과정(test process)을 용이하게 하는 성질들을 갖고 있다는 것을 의미한다[2]. 소프트웨어 시험성을 향상시킬 수 있는 메커니즘들 중의 하나가 테스트 오러클(test oracle)이다.

테스트 오러클은 테스트 실행에 대한 허용 가능한 행위(acceptable behavior)를 규정하여 시험 대상 소프트웨어가 명세에 명시된 기능을 올바로 수행하는지를 판단할 수 있는 수단을 제공한다[3, 4].

소프트웨어의 시험에 테스트 오러클이 제공된다면 해당 SUT의 시험성은 향상될 수 있다. 반면에, SUT의 시험 결과를 오러클을 이용하여 판단할 수 없다면 SUT의 고장(failure)을 찾아내거나 올바른 행위를 확인하고자 하는 시험의 본래 목표는 실질적으로 달성되기 어렵다.

테스트 오러클은 일반적으로 수작업을 통해 작성된다. 이로 인해, 테스트 오러클의 관리 비용이 많이 들고 테스트 스위트(test suite)들에 대한 처리 능력도 떨어진다. 뿐만 아니라, 오러클 작성에 사람이 개입함에 따라 오러클에 오류가 내재할 가능성이 있어 오러클이 본래 고장들을 찾지 못하거나 정상적인 결과를 고장으로 비롯된 결과로 오인할 수 있다[1]. 따라서 보다 높은 신뢰성을 갖는 소프트웨어를 개발 위해서는 보다 많은 부분의 테스트 오러클 개발 과정이 자동화되어져야 한다.

테스트 오러클의 자동화는 테스트 오러클 생성 과정의 자동화와 테스트 오러클 실행의 자동화로 구분할 수 있다. 본 연구는 두 가지 측면의 자동화를 지원하는 테스트 오러클 생성 지원 환경의 구축 방법을 제안한다. 본 연구의 목적은 Statechart로 기술된 시험 대상 소프트웨어의 행위 모델로부터 실행 가능한 테스트 오러클을 생성하는 시험 환경을 구축하는 것이다.

2. 기존 연구

본 장에서는 테스트 오러클에 대한 기존 연구 내용들을 간략하게 살펴본다.

[5, 6, 7]은 정형 명세로부터 수동적으로 테스트 오러클을 생성하는 방법들을 제시하였다.

[5]은 Z[10]로 작성된 소프트웨어 명세를 기반으로 하여 자동화 된 테스트 오러클을 구축하는 방법에 대하여 기술하였고, [6]와 [7]은 Object-Z[11]로 정형 명세된 컨테이너(container) 클래스로부터 CUT(Class Under Test: 시험 대상 클래스)의 행위를 흉내내는 active 오러클과 CUT의 행위만을 점검하는 passive 오러클을 각각 도출해 내는 과정을 설명하였다.

[5, 6, 7]과 달리 [8]은 테스트 오러클 뿐만 아니라 전반적인 시험 메커니즘의 구축 방법들에 대하여 기술하였다. [8]은 CUT(Class Under Test: 시험 대상 클래스)의 시험 대상 행위를 State 다이어그램의 일종인 Testgraph로 명세하고, 이를 토대로 테스트 케이스와 테스트 오러클을 생성하여 CUT를 시험하는 방법을 제시하였다. [8]의 테스트 오러클 생성 방법은 상태 머신(state machine) 형태로 기술된 CUT의 행위 모델로부터 시험자가 테스트 오러클을 개발하는 것이다.

3. 소프트웨어 행위 명세와 테스트 오러클

그림 1은 명세, 소프트웨어, Statechart 모델, 그리고 테스트 오러클들 간의 관계를 표현한 클래스 다이어그램이다.

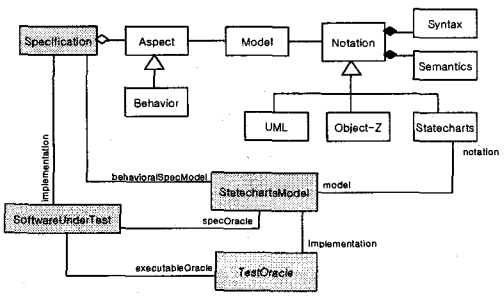

그림 1: 소프트웨어 행위 명세와 테스트 오러클들 간의 관계

명세는 구현될 소프트웨어가 만족해야 하는 사항들을 명시한 문서로 볼 수 있다. 명세는 구현될 소프트웨어가 만족해야 할 사항들을 다양한 측면(aspect)에서 기술한다. 명세는 각 측면들을 모델(model)로써 표현한다.

본 연구는 소프트웨어의 행위적 측면에 초점을 맞춘다.

소프트웨어 모델링을 위해서는 표기법(notation)이 필요하다. 본 연구는 소프트웨어의 행위적인 측면은 Statechart로, Statechart 의미 메타모델은 UML로, 그리고 Statechart 의미 메타모델의 각 구조체는 Object-Z로 명세한다.

Statechart로 기술된 소프트웨어 행위 모델은 SUT에 대한 명세 오러클(*specification oracle*) 역할을 수행하고, 이를 구현한 테스트 오러클(test oracle)은 실행 가능한 오러클(executable oracle) 역할을 수행한다.

3.1 Statechart 의미 메타모델

Statechart[9]는 멀티 컴퓨터 실시간 시스템들, 통신 프로토콜들, 그리고 디지털 제어 장치들과 같은 복잡한 이산-이벤트 시스템들을 명세하고 설계하는데 적절히 사용되었던 종래의 상태 머신(*state machine*)과 상태 다이어그램을 확장한 다이어그램으로서 SUT의 행위 모델을 기술하는데 유용하게 사용할 수 있다.

그림 2은 명세 테스트 오러클로 사용할 Statechart에 대한 의미 메타모델(semantic meta-model)을 나타내는 UML 클래스 다이어그램이다.

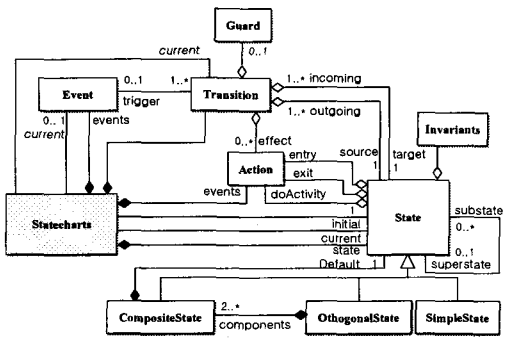

그림 2: Statechart 의미 메타모델(semantic meta-model)

그림 2에서 클래스로 표현된 Statechart 의미 메타모델의 각 구조체를 간략하게 설명하면 다음과 같다.

Statechart는 상태 집합, 이벤트 전이 집합, 액션 집합들로 구성된다.

상태는 0개 이상의 들어오는(incoming) 전이와 0개 이상의 나가는(outgoing) 전이를 갖는다. 그리고 상태는 1개 이하의 슈퍼 상태(superstate), 0 이상의 서브 상태를 갖는다. 어떤 상태로 전이되어 들어갈 때, 어떤 상태에서 전이되어 나갈 때, 또는 어떤 상태에 머무르는 동안 어떤 액션들이 취해질 수 있다.

Statechart의 상태는 Simple 상태, Composite 상태, 그리고 Orthogonal 상태로 구분된다.

Simple 상태는 서브 상태가 없는 상태이다. 따라서 상태 계층 상에서 가장 하위에 있는 상태이다. Composite 상태는 서브 상태(substate)들을 포함하고 있는 상태이다. 그리고 Orthogonal 상태는 병렬 컴포넌

트(parallel component)들을 서브상태로 포함하고 있는 상태이다.

본래 Statechart에 불변조건(Invariant) 개념을 추가하였다. 불변조건은 SUT의 테스트 입력에 대한 출력 결과가 Statechart로 기술된 행위 모델 상의 올바른 어떤 상태로 매핑되었을 때 그 상태에서 SUT의 출력 결과가 항상 만족해야 하는 제약 조건들을 의미한다.

3.2 Object-Z

Object-Z[10]는 Z[11]를 확장한 정형 언어이다. 다시 말해, Object-Z는 기존의 Z에서 명세하기 힘든 객체지향 스타일의 소프트웨어를 쉽게 명세하기 위한 개념들을 추가, 확장한 언어이다. Object-Z에서 확장된 주요 개념들은 클래스 스키마(class schema), 상속(inheritance), 그리고 인터페이스(interface)이다.

Object-Z의 클래스 스키마는 객체지향 개념의 클래스를 표현한다. 상속 메커니즘은 기존의 클래스 스키마들을 재사용하여 새로운 클래스 스키마를 정의할 수 있게 한다. 그리고, 인터페이스는 외부 환경이 접근할 수 있는 클래스 스키마의 상수, 상태 변수, 초기 상태 스키마, 그리고 오퍼레이션들을 선언한다. 따라서, 클래스 객체들은 해당 클래스 스키마의 인터페이스를 통해 가시성(visibility)을 조정할 수 있다.

Object-Z가 위와 같은 메커니즘들을 지원하기 때문에 그림 1에 표현된 Statechart 의미 메타모델의 각 구조체들의 속성, 오퍼레이션, 제약조건들을 Object-Z의 클래스 스키마 내에 정형적으로 명세할 수 있다.

[12]는 UML의 상태 머신(state machine)에 대한 의미 메타모델의 추상 구문(abstract syntax)과 정적 및 동적 의미를 Object-Z로 정형 명세하였다.

본 연구는 그림 1의 Statechart 의미 메타모델의 각 구조체들을 [8]과 유사한 구조를 가지면서도 객체지향 구현 언어로 쉽게 매핑 가능하도록 Object-Z로 명세한다.

4. Statechart 행위 모델 기반의 테스트 오러클 생성 지원 환경

소프트웨어 시험성 강화를 위한 대표적인 장치들은 테스트 케이스 생성기, 테스트 케이스에 대한 데이터의 입력과 초기 조건의 설정을 제어하기 위한 테스트 제어기, 테스트 실행 결과를 감시하는 테스트 감시기, 그리고 테스트 케이스에 기대된 시험의 사전, 사후 및 불변 조건(expected pre-, post-condition, invariant)들을 알려주어 시험의 성공 여부를 판정할 수 있게 하는 테스트 오러클이다.

본 연구는 시험성 강화 메커니즘들 중 테스트 오러클에 초점을 둔다.

그림 3은 본 논문에서 제안한 테스트 오러클 생성 지원 환경에 초점을 맞추어 나타낸 다이어그램이다.

Statechart 모델은 구현될 소프트웨어가 만족해야만 하는 행위적 제약 조건들을 명세한다. 따라서 SUT(Software Under Test: 시험 대상 소프트웨어)는 Statechart 모델에 명세된 제약 조건들을 만족해야 한다.

테스트 오러클(Test Oracle)은 Statechart로 표현된 SUT의 행위 모델로부터 도출된다. 따라서 테스트 오러클은 Statechart 행위 모델을 실행 가능하게 구현한 것으로 볼 수 있다.

일반적으로 Statechart 모델은 SUT의 전체 행위 중 시험 관심 대상인 일부 행위만을 명세한다. 따라서 Statechart 행위 모델을 구현한 테스트 오러클은 SUT보다 추상적이며 작은 상태 공간을 갖는다.

테스트 드라이버(Test Driver)는 그림3에 표현하지 않았지만 테스트 케이스 생성기(Test Case Generator)에게 SUT를 위한 테스트 입력의 생성을 요청하거나 테스트 오러클 생성기(Test Oracle Generator)에게 SUT를 위한 테스트 오러클의 생성을 요청한다. 그리고 테스트 드라이버는 생성된 동일한 테스트 입력을 시험 대상 소프트웨어(Software Under Test)와 실행 가능한 테스트 오러클(Test Oracle)에게 전달한다.

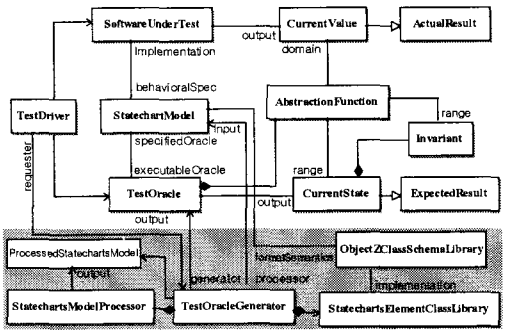

그림 3: Statechart 행위 모델 기반의 테스트 오러클 생성 지원 환경

테스트 드라이버가 전달한 테스트 입력에 대하여, SUT는 실제 출력 결과(Actual Result) 즉, 현재 값(Current Value)을 출력하고 테스트 오러클은 기대 출력 결과(Expected Result) 즉, 현재 상태(Current State)를 생성한다. SUT의 현재 값은 SUT의 인스턴스 변수(instance variable)들에 저장된 값들의 조합으로서 표현되고, 테스트 오러클의 현재 상태는 Statechart 모델 상의 현재 상태를 나타낸다.

테스트 오러클의 출력은 현재 상태에서 SUT의 출력이 만족해야만 하는 불변조건(invariant)을 포함하고 있다.

대개, 명세의 값 공간과 구현된 소프트웨어의 값 공간이 다르다. 따라서 실행 가능한 행위 모델라고 볼 수 있는 테스트 오러클의 출력 값 공간과 SUT의 출력 값 공간 간의 매핑 정의해주는 메커니즘이 필요하다[3]. 추상 함수(Abstract Function)가 이와 같은 역할을 수행한다.

본 연구에서, 추상 함수(Abstract Function)는 1) 테스트 입력에 대한 SUT의 출력 결과 즉, 현재 값들을 테스트 오러클 상의 값 공간(value space) 즉, 현재 상태로 변환해 주는 역할과 2) 테스트 입력에 대한 SUT의 출력 결과를 불변조건의 값 공간으로 변환하는 역할

을 수행한다. 불변 조건은 테스트 오러클의 현재 상태에서 만족해야 제약 조건들이다.

지금까지, Statechart로 기술된 SUT의 행위 모델로부터 생성한 실행 가능한 테스트 오러클을 이용하여 SUT의 실행 결과를 점검하는 과정을 기술하였다.

이제는, 테스트 오러클 생성 지원 환경의 각 구성 요소에 대하여 간략하게 설명한다. 그림 3의 음영 부분이 테스트 오러클 생성 지원 환경의 각 구성 요소들을 나타낸다.

Statechart 모델로부터 실행 가능한 테스트 오러클을 생성하기 위해서는, 우선 다이어그램 형태의 Statechart 모델을 동일한 의미를 갖는 텍스트 형태의 Statechart 모델로 변환하여 구문을 검사 한 후, 최종적으로 테스트 오러클 생성기(Test Oracle Generator)가 처리하기 용이한 내부 추상 표현(internal abstract representation)인 메타 객체들의 형태로 가공할 수 있는 장치가 필요하다. 이와 같은 기능은 Statechart 모델 처리기가 수행한다.

Statechart 모델 처리기에 의해 처리된 Statechart 모델(Processed Statechart model)은 자신의 모델 정보를 테스트 오러클 생성기가 접근할 수 있도록 인터페이스를 제공한다.

Statechart 요소 클래스 라이브러리(Statechart Element Class Library)는 그림 1에서 표현한 Statechart 의미 메타모델의 각 구조체들을 구체적으로 구현한 클래스들의 모임이다. Statechart 요소 클래스 라이브러리를 구성하는 각 클래스에 대한 속성, 오퍼레이션, 인터페이스, 그리고 제약조건들을 해당 Object-Z 클래스 스키마 내에 정형적으로 명세한다.

테스트 드라이버가 테스트 오러클 생성기에게 SUT에 대한 테스트 오러클 생성을 요청하였을 때, 테스트 오러클 생성기는 다음과 같은 과정을 수행한다.

① 테스트 오러클 생성기는 Statechart로 기술된 SUT의 행위 모델을 입력으로 받는다.

② 테스트 오러클 생성기는 입력 받은 SUT의 행위 모델 즉, Statechart 모델을 자신의 내부 컴포넌트로 포함하고 있는 Statechart 모델 처리기에게 의미적으로 동일하지만 테스트 오러클의 생성 시 쉽게 사용할 수 있는 형태로 변환해 줄 것을 요청한다.

③ Statechart 모델 처리기는 Statechart로 기술된 SUT의 행위 모델을 구문 검사한 후, 이를 테스트 오러클 생성기가 처리하기 쉬운 형태의 내부 추상 모델로 변환한다.

④ 테스트 오러클 생성기는 Statechart 모델 처리기의 산출물을 입력으로 받아, Statechart 요소 클래스 라이브러리에서 제공하는 클래스들을 이용하여 Statechart 행위 모델에 대응하는 실행 가능한 테스트 오러클을 생성한다. 생성된 테스트 오러클은 Statechart 요소 클래스 라이브러리에서 제공하는 클래스들의 객체들로 구성된다.

5. 결론 및 향후연구

본 논문은 소프트웨어의 동적인 측면을 기술하는데 유용한 Statechart로 기술된 소프트웨어 행위 모델로부터 실행 가능한 테스트 오러클을 생성하여 SUT의 시험성을 향상시킬 수 있는 시험 환경의 구축 방법을 제안하였다.

현재, Statechart 행위 모델을 테스트 오러클로 사용하기 적합하도록 Statechart 의미 메타모델을 정제하는 동시에, Object-Z를 이용하여 Statechart 의미 메타모델의 각 구조체들을 객체지향 프로그래밍 언어의 클래스로 쉽게 매핑시켜 구현할 수 있도록 정형화하고 있다.

향후에는, 현재 진행 중인 연구들을 기반으로 하여 Statechart 모델 처리기와 테스트 오러클 생성기 부분으로 크게 구분하여, 본 논문에서 제안한 테스트 오러클 환경을 개발해 나갈 계획이다.

참고문헌

[1] Douglas Hoffman, "A Taxonomy of Test Oracles," Quality Week 1998, Software Qaulity Methods LLC.

[2] S. T. Chanson and A. F. Loureiro and S. T. Vuong, "On the Design for Testability of Communication Software," Proceedings of IEEE International Test Conference, 1993, pp. 190-199.

[3] D. J. Richardson, S. L. Aha, T. O. O'Malley, Specification-based Test Oracles for Reactive Systems," International Conference on Software Engineering, 1992. 1992, pp. 105–118.

[4] J. McDonald, P. Strooper, "Translating Object-Z specifications to passive test oracles," Second International Conference on Formal Engineering Methods, 1998, pp. 165 – 174.

[5] Luqi, Hongji Yang, Xiaodong Zhang "Constructing an automated testing oracle: an effort to produce reliable software,"Computer Software and Applications Conference, 1994. COMPSAC 94. Proceedings., Eighteenth Annual International , 1994 , pp. 228 -233

[6] J. McDonald, L. Murray, P.Strooper, "Translating Object-Z specifications to object-oriented test oracles," Asia Pacific Software Engineering Conference 1997, and International Computer Science Conference 1997, pp. 414-423.

[7] J. McDonald, P. Strooper, "Translating Object-Z specifications to passive test oracles," Second International Conference on Formal Engineering Methods, 1998, pp. 165 - 174

[8] D. Hoffman and P Strooper, "ClassBench: a Framework for Automated Class Testing", Software Maintenance: Practice and Experience, Vol. 27, No. 5, May 1997, pp. 573-597.

[9] D. Harel, Statecharts: A Visual Formalism for Complex Systems", Science of Programming, Vol. 8, 1987, pp. 231-274.

[10] Graeme Smith, "The OBJECT-Z Specification Language. Advances in Formal Methods", Kluwer Academic Publisher, 2000.

[11] J. M. Spivey, The Z Notation: A Reference Manual, 2nd. Edition, 1992.

[12] Soon-Kyeong Kim and David Carrington, A Formal Model of the UML Metamodel: The UML State Machine and Its Integrity Constraints," LNCS 2272, 2002, pp. 497-526.