복잡한 내장형 소프트웨어를 위한 테스트 모델 설계

이명호[1], 이상용[2], 장중순[2], 최경희[3], 박승규[3], 정기현[4]
[1]세명대학교 인터넷정보학부 / [2]아주대학교 산업정보시스템공학부
[3]아주대학교 정보 및 컴퓨터공학부 / [4]아주대학교 전자공학부

Design of Testing Model for Complex Embedded Software

M.H. Lee, S.Y. Lee, J.S. Jang, K.H. Choi, S.K. Park, K.H. Jung
Div. of Internet Information, Semyung Univ. / Industrial Eng., Ajou Univ. / Computer Eng., Ajou Univ. / Electronics Eng., Ajou Univ.

Abstract
As information telecommunication industry develops and systems get integrated gradually, the importance of embedded software is growing significantly. On the contrary, the reliability of embedded software has worsened. Accordingly, testing of software is an essential part in designing and building up embedded system and the volume of testing required is immense.

This paper elaborates on design and testing model for complex embedded software, providing guidelines to collect requirements for complex embedded software.

Keywords : *complex embedded software, design, testing model*

1. 서론

정보통신 산업의 발전과 점진적인 시스템들의 통합화에 따라 내장형 소프트웨어의 중요성은 더욱 커지고 있다. 또한 첨단기술 산업이 발전함으로 복잡한 내장형 시스템의 수요는 모든 분야에서 점차 증가하고 있는 추세지만 내장형 소프트웨어의 신뢰성은 점점 떨어지고 있다. 그러나 사전 조건 데이터 및 상태, 온도, 습도, 광학 같은 환경 조건을 가진 내장형 시스템에 대한 모델 표현이나 테스트를 하고자 하는 시스템의 대상 내에서 센서나 액츄에이터와 같은 제어 로직이 내장된 복잡한 내장형 소프트웨어의 테스트 모델에 대한 연구는 현재까지 미비한 실정이다. 내장형 소프트웨어의 테스트에서 가장 어려운 문제 중 하나는 타당성 있고 신뢰성 있는 테스트 자료 생성 전략을 수립하는 것이다.

그동안 프로그램의 유형 분석 및 통신 프로토콜 등을 모델링 하는데 FSM(Finite State Machine)으로 많은 표현이 되어 왔다. 이른바 FSM은 내장형 소프트웨어의 상태기반 행동특성(state-based behavior)을 이용하여 테스트 케이스를 생성하고자 하는 것이다. 또한 시스템 행동특성의 관점을 발견하기 위하여 FSM으로 테스트 문제 연구에 동기를 부여하였다(David Lee, 1996).

일반적으로 많은 내장형 시스템이나 부분들은 상태기반 행동특성을 보이기 때문에 시스템들을 설계하는데 상태기반 모델링이 사용된다. 이러한 과정 동안에 구성된 모델들은 테스트 설계에 대한 기초로서 제공된다. 상태기반 테스트는 이벤트(events), 액션(actions), 활동(activities), 상태(states), 그리고 상태 전이(state transitions)간의 연관관계를 확인하기 위한 것이다. 그러나 FSM을 사전 조건 데이터 및 환경 조건을 가진 복잡한 내장형 소프트웨어 시스템에 적용 시에는 상태기반 행동특성 및 상태 전이의 연관관계를 잘 표현해 주지 못하는 단점이 있다.

따라서 본 논문에서는 기존의 모델링 방법을 기반으로 하여, 복잡한 내장형 소프트웨어 시스템을 위한 상태기반 모델을 설계하고, 이 설계를 기반으로 복잡한 내장형 소프트웨어 테스트(*CEST*:Complex Embedded Software Testing) 모델을 표현하도록 한다. 또한 복잡한 내장형 소프트웨어 테스트 구조를 설계하여, 향후 내장형 소프트웨어의 테스트를 위한 기본 지침 제공과 내장형 소프트웨어를 위한 테스트 요구사항을 수집하는데 도움을 줄 수 있도록 한다.

[2]이상용, 수원시 영통구 아주대학교 팔달관 229 FAX:031-219-1610
E-mail:biztech.ajou.ac.kr

2. FSM 모델

FSM이란 어떤 모델이 출발 상태, 입력 알파벳 그리고 입력 심볼과 현재 상태가 다음 상태로 매핑된 전이 함수(transition function)로 구성된다. 일반적으로 FSM은 다음과 같이 6개의 튜플로 표기할 수 있다.

$$(S, I, O, \delta, \lambda, q_0)$$

여기서 S는 상태들의 유한집합, I는 유한한 입력알파벳, O는 유한개의 출력알파벳, δ 는 $S \times I \rightarrow S$ 인 전이함수, λ는 출력함수, q_0는 S의 원소인 시작 상태 혹은 최초 상태(start state, initial state)이다. 그 중에서도 출력함수에 따라, 출력이 현재 상태에만 의존되는 무어 머신(moor's machine)은 $\lambda : S \rightarrow O$ 인 출력함수가 되며, 출력이 현재 상태와 입력에 의존하는 밀리 머신(mealy's machine)은 $\lambda : S \times I \rightarrow O$ 인 출력 함수가 된다.

그러나 위와 같은 튜플을 가진 FSM 모델 표현으로는 같은 입력으로 사전 조건 데이터 및 환경 조건에 따라 다른 출력을 나타낼 수 있는 복잡한 내장형 소프트웨어 시스템 모델을 표현할 수가 없다.

3. 복잡한 내장형 소프트웨어 모델의 설계

3.1 모델의 표현

복잡한 내장형 소프트웨어 중 시작 상태를 알고 있으면서 전이가 일어날 때, 이전 상태의 환경 조건, 사전 조건 가드 및 이벤트 조건에 따라 다음 상태에 종속되는 모델일 경우에 다음과 같이 5개의 튜플(tuple)과 하부 전이 함수에서 다시 5개 튜플을 가진 내부 전이 함수(Inner Transition Function)로 표현할 수 있다.

$$CEST = (\Theta, \Sigma, \Delta, \xi_0, \Phi)$$

여기서 Θ는 상태(node)들의 유한집합, Σ는 유한한 이벤트(event)들, ξ_0는 Θ의 원소인 시작상태 혹은 최초상태(start state, initial state), Φ은 leaf 상태들의 집합, Δ는 전이함수(transition function)이며,Δ인 전이함수는 다음과 같이 5개의 튜플로 표현된다.

$$\Delta = (\Theta_s, preC, \epsilon, posC, \Theta_d)$$

여기에서 $\Theta_s (\Theta_s \in \Theta)$와 $\Theta_d (\Theta_d \in \Theta)$는 특히 source state와 destination state를 표현한다. $preC(preC \subset \Theta)$와 $posC(posC \subset \Theta)$는 사전조건(precondition)과 사후조건(postcondition)을 표현한다. $\epsilon(\epsilon \in \Sigma)$는 Θ_s 상태와 관련된 event를 표현한다. $preC$와 posC는 입력(incoming)과 출력(outgoing) 데이터를 포함한다.

Test Scenario를 표현하는 initial node ξ_0에서부터 leaf node $\phi(\phi \in \Phi)$까지의 각 경로는 순서화된 이벤트들의 시간 시퀀스와, 사전/사후 조건들과 목록, 입력 데이터와 출력 데이터(출력 기대값)의 목록 등이다.

어떤 테스트 시나리오(Test Scenario)는 다음과 같은 순서 집합으로 정의된다(W.T. Tsai et al., 2003).

$$\Psi = (\Psi_i \mid (\xi_0, \delta_{i1}, \theta_{i1}, \cdot \cdot \cdot, \Phi), \delta_{i1} \in \Delta, \theta_{i1} \in \Theta, i = 1, 2, \cdot \cdot \cdot, n)$$

여기에서 n은 테스트 시나리오의 총 개수이다.

3.2 테스트 모델의 설계

소프트웨어의 테스트하기 위해서는 테스트케이스를 통한 시스템의 결과와 소프트웨어 스펙이 정의하고 있는 결과, 즉 기대치를 비교하는 것이다. 이러한 기대치는 테스트 오라클(Test Oracles)이라 불리는 매카니즘을 이용하여 생성된다. 오라클의 단어는 테스트에서 몇 가지 의미로 사용된다. 기대치를 생성하는 과정, 기대치 그 자체, 혹은 실제 결과가 우리의 기대값인지 아닌지에 대한 대답 등이다. 이 항목에 대하여, 오라클이라는 단어는 기대치를 생성하는데 사용되는 다른 프로그램이나 매카니즘을 의미하는데 사용된

Table 1. 5 Approaches to Oracles

항목	True Oracle	Heuristic Oracle	Sampling Oracle	Consistent Oracle	No Oracle
정의	▪ 모든 기대치의 독립적인 생성	▪ 나머지 값의 일치성뿐만 아니라 어떤 값들을 검증한다.	▪ 입력이나 결과의 특별한 모집을 선택한다.	▪ 현재 실행 결과를 이전에 실행 결과(회귀분석테스트)로 검증한다.	▪ 결과의 정확성을 체크하지 못한다.
장점	▪ 감지된 에러가 탐지되지 않는 것은 어떠한 것도 없다.	▪ True Oracle보다 빠르고 쉽다. ▪ 생성하고 사용하는데 비용이 많이 들지 않는다.	▪ 쉬운 계산된 값이나 인지된 결과들을 선택할 수 있다. ▪ 단지 단순한 오라클을 이용하여 수작업으로 검증할 수 있다.	▪ 오라클을 이용하는 가장 빠른 방법. ▪ 검증이 간단하다. ▪ 데이터의 많은 량을 생성하고 검증할 수 있다.	▪ 데이터의 어떠한 용량도 실행할 수 있다(단지 SUT 작업 시간에 제약됨).
단점	▪ 적용하는데 고비용이 든다. ▪ 복잡하고 실행될 때	▪ 시스템적인 에러를 놓칠 수 있다(다음과 같은 Sine 곡선 사례처럼).	▪ 시스템적으로 특화된 에러를 놓칠 수 있다. ▪ 종종 “소프트웨어가 테스	▪ 최초의 실행은 감지되지 않은 에러를 포함할 수 있다.	▪ 단지 극적인 실패가 나타난다.

종종 시간이 많이 소비된다.

다. 따라서 경험적 접근에 관한 결정은 테스트 상황에 따라 하는 것이 최상의 방법이다.

Table 1은 자동화된 소프트웨어 테스트를 검증하기 위한 소프트웨어 산업에 따라 성공적으로 채택되었던 오라클에 대한 5가지 접근에 대하여 장·단점을 기술한 것이다 (Douglas Hoffman , 1999).

일반적으로 내장형 시스템의 테스트 모델은 테스트 입력을 주고 테스트를 하고자 하는 시스템의 대상(System Under Test)은 Black Box 형태를 통하여 테스트 출력이 이루어지는 모델이다. 일반적인 Black Box 테스트 모델을 도식화해 보면 <그림 1>과 같다.

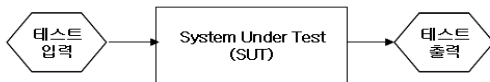

Figure 1. General Black Box Test Model

그러나 위와 같은 테스트 입력이외에 사전조건 데이터 및 환경 조건이 있으면서 테스트를 하고자 하는 소프트웨어의 대상 내에 센서나 액츄에이터와 같은 제어 로직이 내장된 복잡한 내장형 소프트웨어의 테스트를 설계하기는 어려운 모델이다.

본 논문에서는 환경 조건뿐만 아니라 테스트를 하고자 하는 시스템의 소프트웨어 대상 내에 제어 로직이 포함되는 복잡한 내장형 소프트웨어의 테스트 모델을 <그림 2>와 같이 설계하도록 한다.

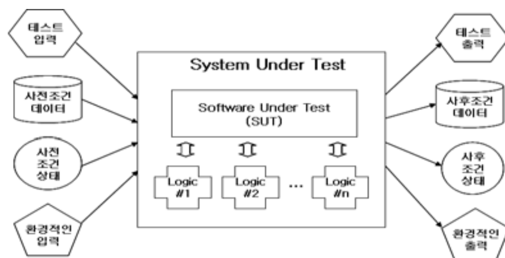

Figure 2. Complex Embedded Software Testing Model

따라서 위에서 언급된 조건들과 오라클을 이용하여 복잡한 내장형 소프트웨어 테스트 모델을 설계해 보면 <그림 3>과 같이 나타낼 수 있다.

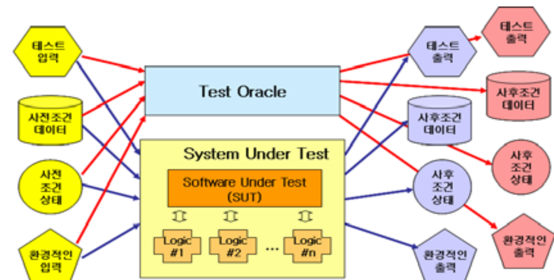

Figure 3. Complex Embedded Software Testing Model with Oracle

3.3 테스트 모델의 구조 설계

내장형 소프트웨어의 테스트 구조 설계는 다음 두 가지 부분으로 설명할 수 있다. 첫째는 테스트 구조 간에 요소들과 연결을 보여주는 구조적인 설명이다. 이것은 데이터 흐름도와 같이 정보를 보여줄 수 있고, 제어와 이동을 묘사할 수 있다. 두 번째 설명은 시퀀스나 이벤트의 흐름이다. 어떤 이벤트가 어떤 과정으로 언제 발생 하는가 이며, 테스트 시퀀스에 대한 과정을 설명하는 부분이다. 모든 단계가 자동화될 필요는 없으며, 항상 비용의 양면성을 고려해야 한다.

이상과 같은 구조 설계를 기반으로 복잡한 내장형 소프트웨어 테스트의 구조는 <그림 4>와 같이 설계토록 한다.

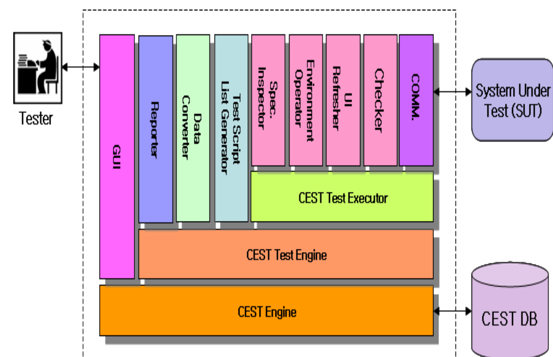

Figure 4. Architecture of Complex Embedded Software Testing

위의 구조에서 CEST 엔진은 전체 모듈을 관리 및 테스트하는 주 엔진 부분이다. GUI (Graphic User Interface)는 사양이나 테스트 시나리오에 따른 기술을 테스터의 입력을 받아 구성되는 모듈이다. CEST 테스트 엔진 부분은 GUI에서 입력 받은 테스트 정보를 기반으로 테스트를 관리하는 모듈이다. 데이터 컨버터는 GUI를 테스트 엔진이 이해할 수 있는 데이터 구조로 변환하는 모듈이다. 테스트 케이스 생성기는 GUI에서 입력 받은 테스트 정보를 이용하여 테스트 스크립트 리스트를 생성하는 모듈이다. CEST 테스트 실행기는 테스트 스크립트 리스트를 테스트 명령어 리스트로 변환시키는 모듈이다. 다시 말하면, 사양 검사기(spec. inspector), 체커(checker), 통신, 사용자 인터페이스 재조작(UI refresh) 모듈에 전달할 명령어로 변경된다. 이 같은 모듈은 테스트 스크립트 리스

트의 양과 수행능력을 최적화시켜주기 위한 설계 방안이다. 마지막으로 CEST 테스트 엔진의 출력기(Reporter)가 실행되어 하나의 테스트 스크립트가 종료된 후 결과를 기록한다.

4. 결론

복잡한 내장형 소프트웨어의 테스트 문제 해결에는 비용과 시간 등의 자원이 방대하게 필요하다는 것이 알려져 있다. 따라서 테스트에서 가장 어려운 문제 중 하나는 타당성 있고 신뢰성 있는 테스트 자료 선택 전략을 발견하는 것이다. 또한 복잡한 내장형 소프트웨어의 모델 설계 연구는 현재까지 미비한 실정이다. 본 논문에서는 기존의 모델링 설계 방법을 기반으로 하여, 복잡한 내장형 소프트웨어를 위한 상태 기반 모델링을 설계하고, 이 설계를 기반으로 복잡한 내장형 소프트웨어의 모델을 표현하도록 한다. 그리고 구조 설계를 통하여 복잡한 내장형 소프트웨어의 테스트를 위한 기본 지침 제공과 테스트 요구사항을 수집하는데 도움을 줄 수 있도록 한다. 향후 이러한 모델의 설계를 기초로 하여 단위 테스트에서 점진적으로 시스템 테스트를 실현할 수 있는 자동화된 내장형 소프트웨어 테스트 도구를 개발하는 것이다.

Reference

Andre'C. Coulter(1999), Graybox Software Testing Methodology : Embedded Software Testing Technique, *Proc. 18th Digital Avionics Systems Conference,* Vol. 2, 10.A.5-1-10.A.5-8.

Boris Beizer(1995),"Black-Box Testing," John Wiley & Sons, Inc.

Bart Broekman, Edwin Notenboom(2003), Testing Embedded software, Addison-Wesley.

David Lee(1996), Principles and Methods of Testing Finite State Machines -A Survey, *Proceedings of the IEEE*, Vol. 84, No. 8.

Douglas Hoffman(1999), Test Automation Architectures:Planning for Test Automation, *Quality Week.*

Douglas Hoffman(1999), Heuristic Test Oracles, *Software Testing & Quality Engineering*, V1, I2, March/April.

Douglas Hoffman(2001), Using Test Oracles in Automation, *Pacific Northwest Software Quality Conference.*

Kwang-Ting Cheng, Jing-yang Jou(1990), Functional Test Generation for Finite State Machines, *International Test Conference*, IEEE, 162-168.

Krishan Sanbani, Anton Dahbura(1988), A Protocol Test Generation Procedure, *Computer Networks and ISDN Systems*, Vol.15, No.4, 285-297.

Tsun S. Chow(1978), Testing Software Design Modeled by Finite-State Machines, *IEEE Transactions on Software Engineering*, Vol. SE-4, No. 3, 178-187.

W. T. Tsai, Tu, X.X. Liu, A, Saimi, Y.Xiao(2003), Scenario-Based Test Case Generation for State-Based Embedded Systems, *Proc. of the 2003 IEEE International Performance, Computing, and Communications Conference,* 335-342.