아키텍처 기술로부터 도출한 의존성 정보를 이용한 테스트 케이스 생성

허유진[0] 오승욱 서희석 권용래
한국과학기술원 전산학과
(yjhuh, suoh, hsseo, kwon)@salmosa.kaist.ac.kr

Test Case Generation using Dependency Information derived from Architecture Description

Yu-Jin Huh[0] Seung-Uk Oh Heui-Seok Seo Yong-Rae Kwon
Dept. of Computer Science, KAIST

요 약

컴포넌트기반 소프트웨어의 신뢰도를 높이기 위해서 개별 컴포넌트의 테스팅과 더불어 통합된 컴포넌트들이 올바르게 동작하는지 테스팅할 필요가 있다. 이 논문에서는 컴포넌트기반 소프트웨어에서 컴포넌트의 구성과 통신에 관한 정보를 얻기 위해 아키텍처 기술을 이용한다. 우선 아키텍처 기술에서 제어 의존성과 자료 요소를 식별하여 테스트 모델을 구축한다. 구축한 테스트 모델에서 테스트의 단위가 되는 OFU(Observable Functional Unit)을 식별하고 자료 의존성을 이용하여 OFU의 수행 순서를 결정한다.

1. 서론

컴포넌트기반 소프트웨어 개발기법은 이미 존재하는 컴포넌트들을 조립함으로써 규모가 큰 시스템을 단시간에 구축하는 기법이다. 이때 개별 컴포넌트의 신뢰도가 높고 컴포넌트들이 올바르게 통합되었을 때 조립된 소프트웨어의 신뢰도가 높아진다. 따라서 컴포넌트기반 소프트웨어의 품질 향상을 위해서는 개별 컴포넌트 뿐 아니라 컴포넌트기반 소프트웨어를 테스팅할 필요가 있다.

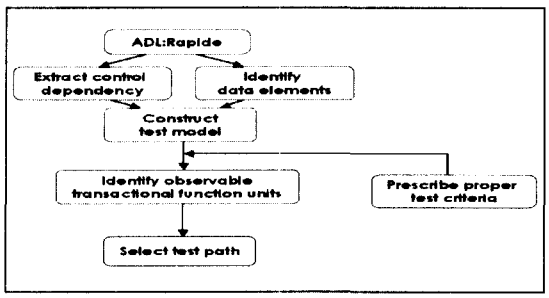
컴포넌트기반 소프트웨어의 테스팅은 컴포넌트의 개발자와 사용자의 관점으로 나눌 수 있다. 컴포넌트 개발자는 컴포넌트가 사용될 환경(context)을 모르기 때문에 모든 가능한 컴포넌트들의 구성(configuration)과 컴포넌트간의 통신(communication)에 대해 테스팅해야 하는데 이는 현실적으로 불가능하다. 그러나, 컴포넌트를 사용하여 컴포넌트기반 소프트웨어를 구축하는 사용자 입장에서는 컴포넌트가 수행될 환경에 대한 정보를 알 수 있으므로 컴포넌트의 행위를 전체 환경에 맞게 테스팅할 수 있다.[1]

Ariane 4 탐사선에서 잘 운용되던 컴포넌트를 Ariane 5 탐사선에 특별한 주의 없이 재사용한 것이 Ariane 5의 폭발의 원인이 되었던 예에서도 알 수 있듯이[2], 컴포넌트를 사용자 관점에 테스팅하는 것은 필수적이다. 컴포넌트 사용자는 일반적으로 컴포넌트의 소스코드에 접근할 수 없고, 아키텍처 기술(architecture description)은 테스팅에 필요한 컴포넌트 구성과 통신에 관한 정보를 잘 표현하고 있다. 이 논문에서는 컴포넌트기반 소프트웨어를 테스팅하기 위해 아키텍처 기술을 기반으로 테스트 케이스를 생성하는 방법을 제안하고자 한다.

아키텍처 기술 언어(Architecture Description Laguage : ADL)에는 Rapide, WRIGHT, Darwin, ACME, UniCon등 여러 가지가 있지만 본 연구에서는 컴포넌트의 구성, 컴포넌트 내부의 추상화된 행위 및 컴포넌트간의 통신을 잘 기술하는 Rapide를 대상으로 한다.

본 연구에서는 아키텍처 기술을 이용한 테스트 케이스 생성 프레임웍을 개발함으로써 컴포넌트기반 소프트웨어의 테스팅을 위한 체계적인 방법을 제시한다. [그림 1]은 테스트 케이스 생성을 위한 전체적 절차를 나타내고 있다. 먼저 대상으로 하는 ADL인 Rapide로 표현된 아키텍처 기술을 프레임웍에 입력한다. 그 다음에 아키텍처 기술로부터 제어 의존성을 추출하고 자료 요소를 식별하여 테스트 모델로 표현한다. 테스트 모델에 규정한 테스트 기준(test criteria)을 적용하여 테스트 수행의 단위를 식별하고 그 단위간의 순서를 결정하여 테스트 케이스의 수행 순서를 결정한다.

이 논문의 구성은 다음과 같다. 2장에서 Rapide와 컴포넌트기반 소프트웨어를 테스팅에 관한 연구를 살펴본다. 3장에서 아키텍처 기술의 분석 및 테스트 모델 구축에 대해 설명하고 4장에서 테스트 모델을 기반으로 테스트 케이스를 생성 방법에 대해 설명한다. 마지막으로 5장에서 결론과 향후 연구 방향에 대해 살펴보도록 한다.



[그림 1] 테스트 케이스 생성 과정

2. 관련연구

2.1 Rapide

D. Luckham이 제안한 Rapide는 컴포넌트의 인터페이스와 외부에서 관찰 가능한 행위를 모델링하는 ADL이다.[3] Rapide는 크게 컴포넌트 기술과 연결관계 기술로 나뉜다. Rapide에서 이벤트란 컴포넌트가 외부와 통신하는 포트(port)의 호출을 의미한다. 컴포넌트 기술은 입력/출력 이벤트들과 이벤트가 발생했을 때의 컴포넌트 행위를 표현하며, 연결관계 기술은 한 컴포넌트의 출력 이벤트들이 다른 컴포넌트의 입력 이벤트들을 발생시키는 규칙을 표현한다. [그림 2]는 Coffee Vending Machine(CVM)의 컴포넌트 기술이며 action부분에서는 외부와 통신하는 이벤트들이, behavior부분에서는 컴포넌트 행위가 기술되어 있다. 예를 들어 [그림 2]의 사각형 부분은 Activate변수가 True일 때 B_milk 이벤트가 발생하면 milk_sugar(Mi) 이벤트가 발생함을 의미한다. [그림 3]는 CVM의 컴포넌트 연결관계 기술이며 사각형 부분은 Button_controller 컴포넌트의 B_milk_sugar 이벤트가 발생하면 Coffee_maker 컴포넌트의 C_milke_sugar 이벤트가 발생함을 의미한다.

```
type Button_controller is interface
action in B_on( ), B_milk( ), B_sugar( ), B_refund( );
       out B_milk_sugar(Mi : Dollars),
           B_sugar(S :Dollars), B_O_refund( );
behavior
  Activate : var Boolean := True;
  Mi, S : var dollars;
  action light_on( );
begin
  (?X : Dollars) B_on ||> Activate :=True; light_on;;
  B_milk where $Activate ||> B_milk_sugar(Mi);;
  B_sugar where $Activate ||> B_O_sugar(S);;
  B_refund   ||> B_O_refund;;
end Button_controller;
```

[그림 2] Rapide로 기술한 CVM 컴포넌트 기술

```
architecture CVM( ) return root
is
  M : Money_controller;
  B : Button_controller;
  C : Coffee_maker;
connect
  (?X : Dollars) B.B_milk_sugar(Mi) ||> M.decrease_money(?X);
  (?X : Dollars) B.B_O_sugar(S) ||> M.decrease_money(?X);
  B.B_milk_sugar ||> C.C_milk_sugar;
            .
            . .
end CVM;
```

[그림 3] Rapide로 기술한 CVM 컴포넌트 연결관계 기술

2.2 컴포넌트기반 소프트웨어 테스팅

컴포넌트기반 소프트웨어를 테스팅하기 위해서 Wu는 테스팅 원소를 인터페이스, 이벤트, 문맥 의존성(context dependence relationship), 내용 의존성으로 정의하였고 이들을 추출하기위한 테스트 모델인 Component Interaction Graph(CIG)를 제안하였다.[4] 이 연구에서는 CIG를 소스코드나 컴포넌트 개발자로부터 얻은 정보로부터 구축할 수 있다고 언급하고 있지만 구체적인 방법은 제시하고 있지 않다.

CHAM을 기반으로 한 테스팅 기법은 CHAM에서 상태를 표현하는 solution이라는 요소와 변환 규칙을 적용하여 시스템의 상태를 다 펼친 그래프를 그린다. 이 중 테스팅에 중요한 행위만 남겨두고 그 외의 행위는 숨겨 그래프를 변환하고 다양한 테스트 기준을 적용하여 시험 경로를 생성한다.[5] 이 방법에서는 모든 가능한 상태를 펼치기 때문에 다이어그램이 복잡해지는 문제가 있으며 대상으로 하는 CHAM언어가 아키텍처의 구성을 기술하는 데 부족한 점이 있어 ADL로 분류되지 않는 단점이 있다.

3. 아키텍처 기술의 분석 및 테스트모델 구축

3.1 제어 의존성 추출 및 자료 요소 식별

Stafford와 Wolf는 소프트웨어 시스템을 위한 아키텍처 레벨의 의존성 분석을 위한 도구를 개발하였다.[6] 주어진 아키텍처 기술에서 e1 이벤트의 발생이 e2 이벤트의 발생을 야기시키면 e2는 e1에 의존성을 갖는다고 보고 모든 이벤트 간의 의존성 유무를 체크하는 의존성 테이블을 만든다. 그리고 사용자로부터 하나의 이벤트를 입력 받아 이로부터 시작하여 이 이벤트에 의존적인 이벤트들을 체인(chain)으로 표현한다. 이 연구에 기반하여 한 컴포넌트 기술에 포함된 이벤트 간의 제어 의존성과 컴포넌트 간의 제어의존성을 정의하고 자료 요소에 대하여 추가적으로 고려하였다.

정의 1 (컴포넌트 내의 제어 의존성) 한 컴포넌트 안에서 직접 혹은 간접적으로 발생하는 이벤트는 이 이벤트를 발생시키는 이벤트에 대해 컴포넌트 내의 제어 의존성이 있다

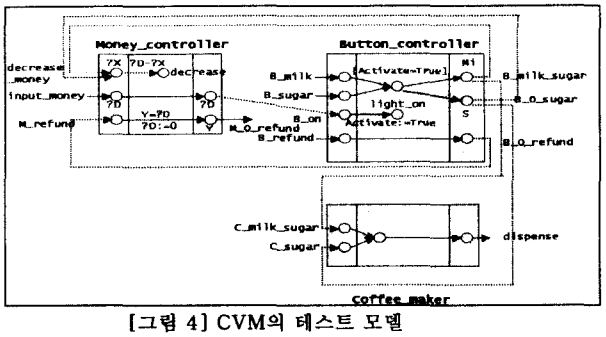
정의 2 (컴포넌트 간의 제어 의존성) 직접 혹은 간접적으로 발생되는 입력 이벤트는 그 입력 이벤트를 발생시키는 다른 컴포넌트의 출력 이벤트에 대해 컴포넌트 간의 제어 의존성이 있다.

[그림 2]의 컴포넌트 기술의 사각형 부분에서 B_milk 이벤트는 B_milk_sugar 이벤트를 직접적으로 발생시키므로 B_milk_sugar 이벤트는 B_milk 이벤트에 대해 컴포넌트 내의 제어 의존성을 갖는다. [그림 3]의 연결관계 기술의 사각형 부분에서 Button_controller 컴포넌트의 B_milk_sugar 이벤트는 Coffee_maker 컴포넌트의 C_milk_sugar 이벤트를 직접적으로 발생시키므로 C_milk_sugar 이벤트는 B_milk_sugar 이벤트에 대해 컴포넌트 간의 제어 의존성을 갖는다.

또한 컴포넌트 내의 행위를 나타내는 규칙에서 정의되고 사용되는 변수에 대한 정보를 자료 요소로 유지한다.

3.2 테스트 모델 구축

3.1에서 추출한 컴포넌트 내의 제어 의존성과 자료 원소를 이용하여 각 컴포넌트에 대한 테스트 모델을 구축하고, 컴포넌트 간의 제어 의존성을 이용하여 컴포넌트 간의 관계를 추가하여 테스트 모델을 완성한다. CVM에 대한 테스트 모델은 [그림 4]와 같다. 각각의 컴포넌트를 사각형으로 표현하고 이를 세 부분으로 나누어 왼쪽부터 입력 이벤트, 컴포넌트 내부 행위, 출력 이벤트를 표현한다. 컴포넌트 내의 제어 의존성을 바탕으로 한 컴포넌트의 각 이벤트간에 에지를 연결하고, 각 에지에 해당되는 자료 요소를 주석으로 표시한다. 컴포넌트 단위로 테스트모델을 구축한 다음, 컴포넌트 간의 제어 의존성을 바탕으로 서로 다른 컴포넌트 간의 이벤트를 에지로 연결하면 [그림 4]와 같은 테스트 모델이 얻어진다.



[그림 4] CVM의 테스트 모델

4. 테스트 케이스 생성

4.1 테스트 단위 식별

정의 3 (Observable Functional Unit : OFU) OFU는 입력 이벤트들이 발생함으로써 야기되는 일련의 이벤트들로 이루어지는 테스트 단위를 말한다. 한 개의 OFU가 끝나면 시스템은 추가적인 입력 이벤트가 발생할 때까지 멈추게 된다.

OFU는 시스템의 입력 이벤트들을 발생시켰을 때 연속적으로 발생하는 이벤트들로 이루어지므로 행위적(behavioral), 기능적(functional) 단위로 볼 수도 있다. OFU는 시스템의 행위와 기능을 테스트하기 위한 최소의 수행단위가 되므로 테스트 단위로 정의한다.

OFU의 식별은 크게 두 단계로 구분된다. 첫 단계에서는 각 컴포넌트 내에서 OFU의 조각(fragment)을 추출하고 두 번째 단계에서는 컴포넌트 간의 연결을 고려하여 조각들을 결합한다.

[그림 4]의 테스트 모델에서 첫 단계로 Button_controller 컴포넌트에서 (B_milk, B_milk_sugar), (B_sugar, B_O_sugar), (B_on, light_on), (B_refund, B_O_refurnd)의 네 개의 조각을 추출할 수 있고, 두 번째 단계로 이들 중 (B_milk, B_milk_sugar) 조각은 컴포넌트 간의 연결에 의해 Money_controller 컴포넌트의 (decrease_money, decrease) 조각과 Coffee_maker 컴포넌트의 (C_milk_sugar, dispense) 조각과 연결되어 밀크커피 버튼을 누르면 밀크커피를 제공하고 돈을 감소시키는 OFU(O1)로 식별될 수 있다. 같은 방법으로 돈을 넣고 CVM에 불이 켜질 때까지의 OFU(O2), sugar버튼을 누른 후 coffee maker에 전달되어 설탕커피가 나오고 돈이 감소되는 데까지의 OFU(O3), refund버튼을 누르고 refund되는 데까지의 OFU(O4)를 식별할 수 있다.

대상이 되는 컴포넌트기반 소프트웨어가 복잡해질수록 OFU의 수가 증가한다. 이때 테스팅할 OFU의 수를 제한하기 위해 테스트 기준을 적용할 수 있다. 적용 가능한 테스트 기준으로는 모든 이벤트의 커버리지(coverage)를 강요하는 all-event criteria, 모든 컴포넌트 간의 연결의 커버리지를 강요하는 all-inter-component-connection criteria등을 고려해 볼 수 있다.

4.2 OFU를 이용한 테스팅 (Testing with OFU)

정의 4 (OFU간의 의존성) 변수 v에 대해서 OFU O1이 v의 정의를 포함하고, OFU O2가 v의 사용을 포함한다면, O2는 O1에 의존성을 갖는다.

OFU를 이용한 테스팅에는 두 가지 형태가 있다. 우선 다른 어떤 OFU에도 의존성을 갖지 않는 OFU는 개별적으로 테스팅할 수 있다. 그러나 다른 OFU O1에 의존성을 갖는 OFU는 O2는 O1과 함께 테스팅 함으로써 추가적인 행위를 테스팅 할 수 있다. 즉 변수 v가 O1에서 정의되고 O2에서 사용되었다면 O2는 O1에 의존성을 가지며 따라서 O1, O2순으로 테스팅할 필요가 있다.

예를 들어 4.1에서 뽑아낸 O1~O4에 대해 OFU간의 의존성을 살펴보자. Button_controller 컴포넌트의 activate변수에 의해서 O1와 O3는 O2에 의존성을 갖는다. 또한 Money_controller 컴포넌트의 D변수에 대해 O1, O3, O4가 O2에 의존성을 갖는다. 돈을 넣어 버튼에 불이 켜지는 O2와 독립적으로 O1과 O3를 테스팅할 때에는 커피가 제공되지 않는 결과를 기대하지만 돈을 넣어 버튼에 불이 켜진 이후에 milk나 sugar버튼을 누르는 순서로 수행하여 커피가 제공되는지 확인하는 것도 의미 있다. 마찬가지로 돈을 넣은 후 refund버튼을 누르는 순서로 수행하는 것도 의미가 있다.

5. 결론 및 향후 연구과제

컴포넌트기반 소프트웨어는 개별 컴포넌트의 신뢰성뿐 아니라 컴포넌트의 통합이 올바르게 되었는지에 따라 신뢰도가 결정된다. 본 연구에서는 아키텍처 기술로부터 배치와 통신에 대한 정보를 추출하여, 통합된 컴포넌트들이 의도한 행위를 보이는지 테스팅하기 위한 테스트 케이스를 생성하는 방법을 제안하였다. 이 방법으로 테스트 단위인 OFU를 식별할 수 있고, 이는 외부 이벤트 입력으로부터 외부 이벤트로의 출력까지 관찰 가능한 기능을 의미하며 시험의 기본 단위가 된다. 또한 기능의 수행 순서를 보여준다. 또한 OFU들의 순서를 정하여 테스트 케이스를 의미 있는 순서로 수행할 수 있게 한다.

향후 연구과제로 아키텍처 기술로부터 의존성 정보를 추출하고 테스트 모델을 만드는 도구와 테스트 모델로부터 OFU를 식별하는 도구를 개발하여 테스트 케이스 생성 과정을 자동화 할 수 있다.

6. 참고 문헌

[1] M. J. Harrold, " Testing : A Roadmap" , *In proceeding of 22nd Int. Conf. on Software Engineering*, June 2000

[2].E. J. Weyuker, " Testing Component-Based Software: A Cautionary Tale," *IEEE Software*, Sep./Oct., 1998

[3] D. Luckham and J. Kenney, " Specification and Analysis of System Architecture using Rapide" , *IEEE Transaction on Software Engineering*, Vol. 21, no 4, pp. 336-354, April 1995

[4] Y. Wu, D. Pan and M. H. Chen, " Techniques for Testing Component-based Software0, *Proceedings. Seventh IEEE Int. Conf. on Engineering of Complex Computer Systems*, 2001.

[5] A. Bertolino, F. Corradini, P. Inverardi and H. Muccine, " Deriving Test Plans from Architectural Descriptions" , On ACM Proceedings, *ICSE2000*, June 2000

[6] J. A. Stafford and A. L. Wolf, " Architecture-level Dependence Analysis in Support of Software maintenance" , *Proceedings of the third international workshop on Software architecture*, 1998, pp. 129-132