UML 상태 다이어그램으로부터 클래스들간 상호 행동의 추출

이우진*, 김영곤**, 김흥남*
*한국전자통신연구원 소프트웨어공학연구부
**한국통신 정보시스템본부
e-mail : woojin@etri.re.kr

Extracting Interclass interactive behaviors from UML State Diagrams

Woo-Jin Lee*, Young-Gon Kim**, Heung-Nam Kim*
*Software Engineering Department, ETRI
**Information System Center, Korea Telecom

요 약

객체 지향 프로그램의 이해 및 테스팅을 효과적으로 수행하기 위해서는 객체 간의 상호 작용을 우선 이해하여야 한다. UML 로 작성된 시스템 명세에서는 각각의 클래스에 대한 행동이 UML 상태 다이어그램으로 기술되어 있어 전체 시스템의 행동을 유추하는데 어려움이 따른다. 이 연구에서는 객체 지향 프로그램의 상태 다이어그램을 기반으로 객체간 행동 테스팅을 수행하기 위해서 UML 상태 다이어그램들을 합성하여 객체간 행동을 추출, 생성하는 과정을 기술한다. 추출, 합성된 객체간 행동 모델은 기존의 널리 알려진 FSM 기반 테스팅 기법들을 그대로 이용할 수 있다.

1. 서론

객체지향 프로그램이 널리 이용됨에 따라 객체지향 프로그램을 체계적으로 테스팅하는 방법이 요구된다. Smith [1]는 객체지향 소프트웨어 테스팅의 4 단계를 제안하였다: 단위 (메소드 단위) 테스팅, 클래스내 (클래스 단위) 테스팅, 클래스간 (클러스터 단위) 테스팅, 그리고 시스템 테스팅이다. 단위 테스팅이나 클래스내 테스팅에 대해서는 기존에 많은 연구가 진행되어 오고 있다. 특히 유한상태머신(FSM)을 기반으로 객체 내부의 행동을 모델링하고 이를 기반으로 테스트 케이스를 생성하는 연구가 많이 진행되었다 [2][3][4][5]. 하지만 클래스간의 행동을 체계적으로 테스팅하는 클래스간의 테스팅에 대한 연구는 많이 진전되어 있지 않다. 클래스간의 테스팅을 지원하기 위해서는 테스팅 모델이 필요한데 객체지향 명세 방법에서는 클래스간의 행동을 명시적으로 나타내지 않고 있어 테스팅 모델을 유추하는데 어려움이 있다.

이 연구에서는 UML [6] 상태 다이어그램으로부터 객체간의 행동만을 추출하여 유한상태머신을 생성하여 클래스간 테스팅의 근간을 마련한다. 클래스간의 관계에는 상속성(inheritance), aggregation, association 등 다양한 관계가 존재하지만 객체간 행위에 직접적으로 연관된 association 관계만을 다룬다. 아래 그림은 이 연구의 전반적인 구조를 보여주고 있다. 먼저 UML 상태 다이어그램 내부에 있는 계층적, 병행적 요소를 제거하고 내부 사건들을 추상화하는 단계를 거쳐 축소된 Extended FSM(EFSM)을 생성한다.

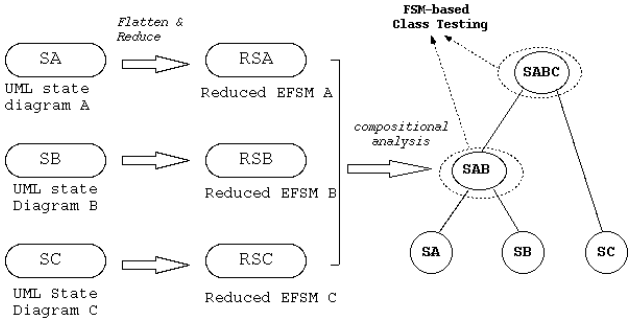
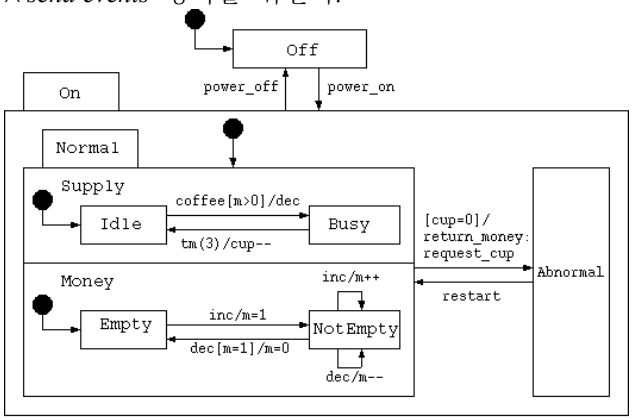


그림 1 : 객체간 행위모델 추출 과정

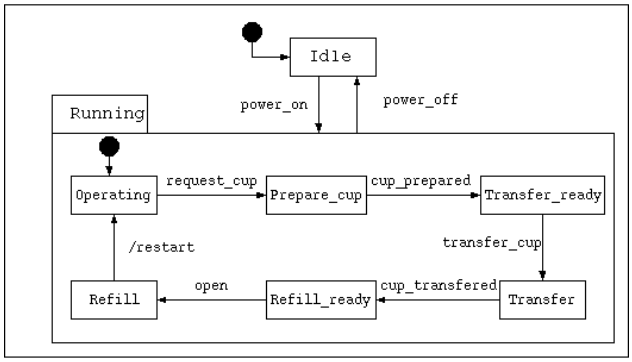
그리고 각 EFSM 들을 계층적으로 합성하여 객체간 행동 모델을 생성한다.

2. UML 상태 다이어그램과 EFSM

UML 에서 클래스의 행동을 기술하는 UML 상태 다이어그램은 계층적인 상태, 병행적 상태, 메시지 방송 통신(message broadcasting communication), 상태와 상태전이에 연관된 행동(actions) 등을 추가적으로 고려한 유한상태머신 (FSM)의 확장이다. 그림 2 는 UML 상태 다이어그램의 예를 보여준다. 그림 2 (a)는 커피 자동 판매기의 행동을 모델링 한 것이며 (b)는 종이컵 유무를 관리하는 제어기 부분을 나타내고 있다. 그림에 나타난 상태전이의 레이블은 *event [guard] / action ∧ send events* 형식을 취한다.



(a) CVM



(b) Controller

그림 2: 커피 자동판매기의 UML 상태 다이어그램

UML 상태 다이어그램에서는 내부구조가 복잡하여 두 개의 클래스 모델을 하나로 합성하는 것이 어려워진다. 그래서 복잡한 내부구조를 단순화시키는 과정이 필요하다. EFSM 은 상태전이에 행동과 메시지 전송을 표현할 수 있도록 확장된 유한상태머신이다. UML 상태 다이어그램에서 EFSM 으로의 변환은 UML 상태 다이어그램의 계층적 상태, 병행적 상태를 제거하는 과정이다. 이러한 평면화 과정은 [5]에서 잘 기술되어 있다. 아래는 EFSM 이 가지는 기본 가정을 나타내고 있다.

- EFSM 의 send events 는 다른 클래스에서 발생하는 이벤트로 구성된다. UML 상태 다이어그램에서는 내부 이벤트들도 send events 에 나타나지만 이러한 이벤트들은 평면화 과정에서 제거된다.
- Guard 부분과 action 부분에 나타나는 상태 변수들은 자료 캡슐화 개념에 따라 다른 클래스와 공유되지 않는다.

그림 3 은 EFSM 의 예를 보여준다. 그림 2(a)에 나타난 UML 상태 다이어그램에서 On, Normal, Supply, Money 의 계층적/병행적 상태를 평면화하여 보여주고 있다.

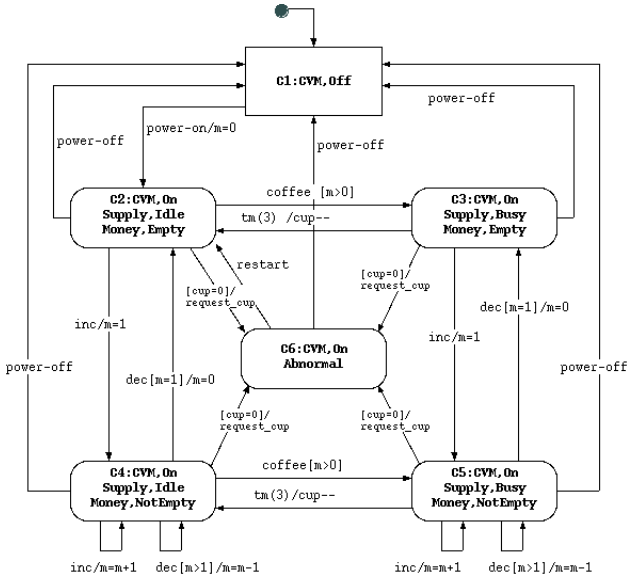


그림 3 : 커피 자동판매기의 EFSM 의 예

3. 내부 구조의 추상화

EFSM 의 내부구조를 축약하기 위해서는 먼저 내부정보와 외부정보를 구분하여야 한다. EFSM 의 각 구성요소에 대해 내부정보의 구분과 추상화에 대해 살펴보자.

- *상태* : 상태 다이어그램에 나타난 상태들은 특별한 의미를 가지지 않으며 또한 다른 클래스에 의해 참조되지 않는다.
- *상태전이* : 상태전이의 내부정보 여부는 상태전이 레이블의 구성요소에 의해 결정된다.
- *이벤트* : 다른 클래스의 send events 에 나타나는 이벤트는 외부 이벤트이며 나타나지 않은 이벤트는 내부 이벤트이다. 내부 이벤트들은 각각의 이름이 의미를 가지지 않으므로 대표 이름인 λ로 바뀌어지며 상태전이는 축약이 가능하다.
- *가드* : 가드 자체는 다른 클래스에 의해 공유되지 않지만 외부 이벤트의 수행순서 결정에 영향을 미칠 수 있다. 즉, 간접적으로 외부 이벤트에 영향을 주므로 외부 이벤트에 영양을 끼치지 않는, 분별력을 상실한 가드에 대해서는 축약이 가능하다.
- *행동(actions)* : 행동은 가드와 비슷한 역할을 가진다. 가드에 사용되는 상태 변수를 가지는 행동(action)은 축약에 신중을 기해야 하며 그렇치 않은 독립적인 상태변수만으로 이루어진 행동은 축약이 가능하다.

- *send 이벤트* : send 이벤트는 다른 클래스에 직접적으로 영향을 미치므로 축약이 불가능하다.

내부 이벤트와 연관된 연속적인 가드와 연속적인 행동들은 하나로 묶어 간략하게 표현할 수 있다. 예를 들어, [x=0]/a=0 와 [y=0]/b=0 의 내부 이벤트로 된 상태전이는 [x=0,y=0]/a=0,b=0 와 같이 하나로 묶어서 표현할 수 있다. 가드와 행동(action)의 축약 가능성은 상태 변수와 밀접한 연관을 가진다. 여러 단계의 상태전이 축약 과정을 거치다 보면 가드에 나타나는 상태 변수의 모든 영역이 다른 상태로의 분기가 발생하지 않고 하나의 상태로 귀결되는 경우가 발생한다. 이러한 경우는 가드의 분별력이 사라지므로 이러한 가드들은 축약이 가능하다. 그리고 분별력이 사라진 가드와 연관된 행동(action) 또한 축약이 이루어진다.

다음은 λ-상태전이 축약 과정을 기술한다. λ-상태전이 축약 과정은 λ-루프(loop)의 제거와 λ-상태전이 제거 과정으로 나뉘어진다. λ-상태전이 축약 과정은 유한상태머신에서 상태전이 축약 방법 [8]을 참조하여 기술한다.

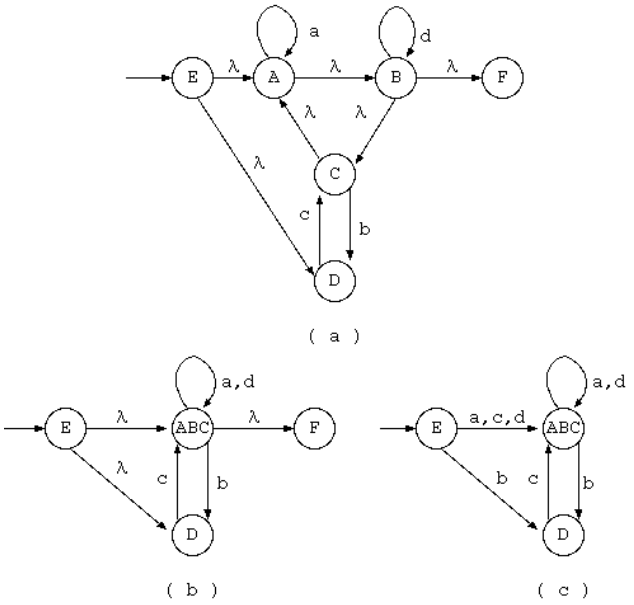


그림 4 : λ-상태전이의 축약

정의 1 *λ-상태전이 루프(loop)의 제거*

하나의 EFSM $M = (GStates, C_0, Gtrans)$이 λ-상태전이 루프를 가진다고 가정하자. 그리고 $States_{lp} \subseteq GStates$ 는 루프에 있는 상태 집합을 나타낸다. 루프가 제거된 λ-accepter $M' = (Gstates', C_0, GTrans)$ 은 아래와 같이 정의된다.

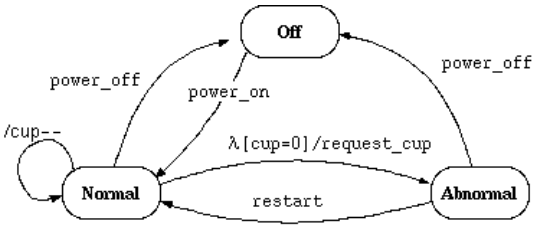
- $GStates' = (GStates - GStates_{lp}) \cup \{ s_{lp} \}$.
- 만약 M 이 $s_1 \xrightarrow{a} s_2, a \in (LABEL \cup \{\lambda\})$이라는 상태전이를 가지면, M'은 아래와 같은 상태전이를 가진다 :

$$s'_1 \xrightarrow{a} s'_2, \text{ where}$$

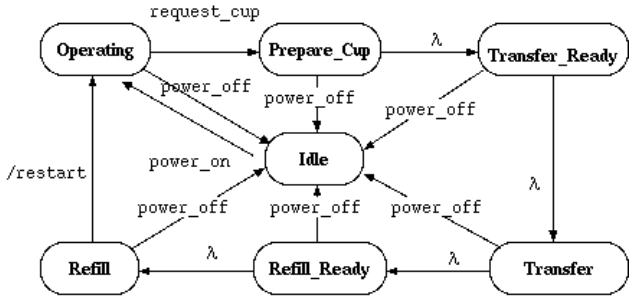
$$s'_1 = \begin{cases} s_{lp}, & \text{if } s_1 \in Gstates_{lp} \\ s_1, & \text{otherwise.} \end{cases}$$

$$s'_2 = \begin{cases} s_{lp}, & \text{if } s_2 \in Gstates_{lp} \\ s_2, & \text{otherwise.} \end{cases}$$

λ-상태전이 루프 제거의 예는 그림 4(a)에 잘 나타나 있다. 그림에 있는 A, B, C 상태로 이루어진 λ-상태전이 루프는 축약과정에서 ABC 상태로 축약되어 그림 4(b)와 같이 된다. 이때 A, B, C 상태 각각으로 입출력되는 상태전이들은 ABC 상태로 입출력되는 것으로 바뀐다.



(a) Class CVM



(b) Class Controller

그림 5 : λ-상태전이 루프의 축약의 예

그림 5(a)는 그림 3 에 나타난 커피 자동판매기 클래스의 EFSM 에서 *inc, dec, coffee* 이벤트를 λ 로 변환한 후 λ-루프를 제거하였다. 그림 5(b)는 종이컵 관리 클래스에 대한 것으로 λ-루프가 존재하지 않아 그림 2(b)의 UML 상태 다이어그램을 EFSM 으로 변환한 모습 그대로 나타나 있다.

λ-상태전이 루프를 제거한 후에 각각의 λ-상태전이들이 제거된다. 연속적으로 이어지는 λ-상태전이들은 하나의 일반적인 이벤트로 구성된 상태전이로 간략화되어 기술된다. 다음은 λ-상태전이 제거 과정을 보여준다.

정의 2 *λ-상태전이의 제거*

$M = (GStates, C_0, GTrans)$ 을 λ-루프가 없는 EFSM 이라 하자. λ-상태전이가 없는 EFSM 인 $M' = (GStates, C_0, GTrans')$ 은 다음과 같이 정의된다:

M' 의 상태전이는 다음과 같다.

- M 에 있는 일반적인 이벤트의 상태전이($s \xrightarrow{a}$

$s', a \in S$)는 M'에 그대로 유지된다.

- $q'' \overset{\lambda}{\Rightarrow} q \xrightarrow{a} q'$. 의 형태를 취하는 상태전이 연속은 하나의 상태전이인 $q'' \xrightarrow{a} q'$ 로 변환된다.

그림 4(b)에 나타난 λ-상태전이들은 모두 변환되어 그림 4(c)와 같이 나타난다. 예를 들어, $E \xrightarrow{\lambda} ABC \xrightarrow{a,d} ABC$ 는 중간과정은 모두 생략되고 처음상태와 끝상태 그리고 이벤트로 이루어진 상태전이인 $E \xrightarrow{a,d} ABC$ 로 변환된다. E→D→ABC, E→ABC→D 도 동일한 규칙에 따라 변환되었으며 ABC→F 는 의미없는 내부 상태 변환이므로 생략된다.

그림 6 은 그림 5 에 나타난 λ-상태전이들을 모두 제거하여 간략히 나타낸 것이다.

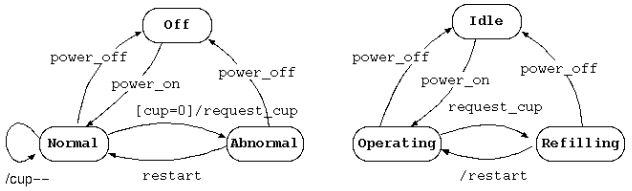


그림 6 : 그림 5 의 λ-상태전이들의 제거

4. EFSM 의 합성

각각의 EFSM 에서 내부적인 구조를 축약한 후, 객체간의 상호작용 정보를 얻기 위해서 축약된 모델들을 공유 이벤트를 중심으로 합성한다. 두 모델의 합성 규칙은 공유되는 이벤트들은 동기화에 의해서만 수행되고 비공유 이벤트는 독립적으로 수행가능하다는 전제 하에 아래와 같이 4 가지 규칙으로 세분화된다. 규칙 (1)은 비공유 이벤트는 다른 모델 수행과 무관하게 독립적으로 수행됨을 보여준다. 규칙 (2)는 send 이벤트가 있는 경우는 두 모델 모두 동일한 이벤트가 발생하여야만 수행이 된다. 규칙 (3)과 규칙 (4)는 규칙 (1)과 규칙 (2)에 가드와 행동을 추가한 것이다. 이러한 합성 규칙들은 CSP 의 합성 규칙 [8]을 응용하여 EFSM 에 적용한 것이다.

$$\frac{P \xrightarrow{e_1} Q, R \xrightarrow{e_2} S}{P \| R \xrightarrow{e_1} Q \| R \ \ or \ \ P \| R \xrightarrow{e_2} P \| S} \quad (1)$$

$$\frac{P \xrightarrow{a/s} Q, R \xrightarrow{s/} S}{P \| R \xrightarrow{a} Q \| S} \quad (2)$$

$$\frac{P \xrightarrow{e_1[g_1]/[a_1]s_1} Q, R \xrightarrow{e_2[g_2]/[a_2]s_2} S}{P \| R \xrightarrow{e_1[g_1]/[a_1]s_1} Q \| R \ \ or \ \ P \| R \xrightarrow{e_2[g_2]/[a_2]s_2} P \| S} \quad (3)$$

$$\frac{P \xrightarrow{e_1[g_1]/[a_1]s_1} Q, R \xrightarrow{s_1[g_2]/[a_2]s_2} S}{P \| R \xrightarrow{e_1[g_1 \wedge g_2]/[a_1 \wedge a_2]s_2} Q \| S} \quad (4)$$

그림 7 은 그림 6 에 나타난 두 모델에 대해 위의 합성 규칙을 적용하여 합성한 모델을 나타내고 있다. 이러한 합성된 모델들은 두 객체간의 상호작용에 연관된 행동만을 모델을 나타내고 있으므로 기존의 FSM 를 기반으로 하는 객체내부 테스팅 방법[5]을 그대로 객체간 상호작용 테스팅에 적용할 수 있다. 테스팅이 완료된 후에는 다른 클래스와 축약, 합성 과정을 반복함으로써 시스템 전반적인 행동을 테스팅할 수 있는 테스트 케이스를 생성할 수 있다.

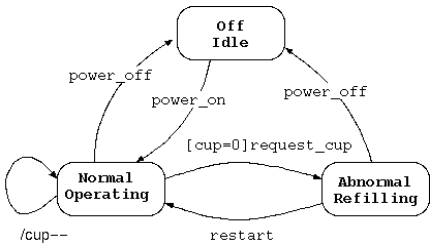


그림 7 : 두 EFSM 모델의 합성

5. 결론 및 향후연구방향

이 논문에서는 객체들간의 상호작용 행동을 합성적으로 생성하는 방법에 대해서 다루었다. 이러한 합성 모델의 생성 방법은 유한상태머신의 직접적인 합성에서 발생할 수 있는 상태 폭발(state explosion)을 회피하면서 적당한 크기의 상호작용 모델을 생성한다. 그리고 합성 모델들을 바탕으로 기존의 FSM 기반 테스팅 방법을 이용하여 객체간 상호작용 테스팅을 수행할 수 있다.

향후 연구 방향으로는 합성적 도달성 트리상에서 테스트 케이스를 생성하는 구체적인 방법과 이들을 지원하는 도구 개발에 관한 연구를 들 수 있다.

참고문헌

[1] C. D. Smith and D. J. Robson, A Framework for Testing Object-Oriented Programs, *Journal of Object-Oriented Programming*, June 1992, 45-53.

[2] D. Kung, N. Suchak, J. Gao, P. Hsia, Y. Toyoshima and C. Chen, On Object State Testing, *Proceedings of Computer Software and Applications Conference*, 1994, 222-227.

[3] C. D. Turner and D. J. Robson, The State-based Testing of Object-Oriented Program, *Proceedings of Conference on Software Maintenance*, 1993, 302-310.

[4] H. S. Hong, Y. R. Kwon, and S. D. Cha, Testing of Object-Oriented Programs Based on Finite State Machines, *Proceedings of Asia-Pacific Software Engineering Conference,* 1995, 234-241.

[5] Y. G. Kim , H. S. Hong, D. H. Bae, and S. D. Cha, Test Cases Generation from UML State Diagrams, *IEE Proceedings-Software*, 146(4), 1999, 187-193.

[6] UML Proposal to the Object Management Group Version 1.1, Rational Corporation, Sep. 1997.(http://www.rational.com/uml/)

[7] W. J. Yeh, *Controlling State Explosion in Reachability Analysis* (Phd. Thesis, Purdue University, Dec. 1993).

[8] P. J. Denning, J. B. Dennis and J. E. Qualitz, *Machines, Languages, and Computation* (Prentice-Hall, 1978).