상호작용 다이어그램을 이용한 설계 컴포넌트 프레임워크의 Use Case 툴 개발

윤태희* · 김영철** · 이재협*
*한국기술교육대학교 정보기술공학부
** 홍익대학교 전자 · 전기 · 컴퓨터공학부 컴퓨터소프트웨어 전공
e-mail : yth00@kut.ac.kr · bob@cic.hongik.ac.kr

Development of Use Case Tool for Design Component Unit Framework based on Interaction Diagram

Tae-Hui Youn* · R. YoungChul Kim** · Jae-Hyub Lee*
*Dept. of Info. Tech. Engineering, KUT.
**Dept. of Computer Software, Hong-ik Univ.

요 약

이 논문은 기존의 Use Case 방법론을 개선하려는데 초점을 두고 있다. Use Case 방법론의 설계단계 중, 설계 프랙티스(Design practices)단계에 소프트웨어 테스팅 개념을 적용하였다. 이 방법은 먼저 사용자의 요구사항을 상호작용 다이어그램을 통해 분석하고, 설계 컴포넌트(Design Component Units)라고 불리는 기능적 컴포넌트들로 새롭게 정의하고 추출한다. 추출된 기반을 바탕으로 생성된 설계 컴포넌트들을 설계 스키마(Design schema)에서 계층구조로 분할하는 방법이다. 여기서 개발자들의 목적에 따라 선택할 수 있도록, 다양한 기준에 적용할 수 있는 여러 가지 설계 컴포넌트들을 정의하였다. 개선된 Use Case 방법론을 토대로 변환 알고리즘[11]을 이용해 새로운 Use Case 툴을 개발하고자 한다.

1. 서론

점차 복잡하고 대형화(그리고 내장화)되는 소프트웨어 추세에, 기존의 개발된 소프트웨어를 배제하고 새로운 소프트웨어를 개발한다는 것은 시간적, 경제적 비용에서 큰 문제가 된다. 이러한 문제를 해결하기 위해 많은 사람들이 컴포넌트 기반 소프트웨어의 재 사용방법을 개발하고 있다. 컴포넌트란 응집력은 강하고 결합력은 약한 소프트웨어의 구현으로, 독립적으로 개발되고 그 자체를 수정하지 않고 다른 컴포넌트와 묶어 새로운 특성을 발생시킬 수 있다[1]. 이러한 컴포넌트들을 재구조화하여 소프트웨어를 개발한다면 많은 비용과 시간을 절감시킬 수 있다. 하지만 컴포넌트로 나누는 기준들이 소스코드 구성레벨, 디자인 레벨, 응용프로그램 레벨, 테스팅 레벨 등 적용되는 레벨단위에 따라 달라지게 되므로, 다른 각도에서 컴포넌트의 기준정의가 필요하다. 여기서 우리는 테스팅 레벨에서 사용할 수 있는 설계 컴포넌트 단위에 초점을 맞추어 Use Case 방법론을 이용, 설계시점에서 상호작용 다이어그램을 통하여 설계 컴포넌트(Design Component Unit)들을 정의하고, 변환 알고리즘[11]을 적용한 Use Case툴에서 설계 컴포넌트들을 찾아내어, 테스트 단계에서 사용하고자 한다.

Use Case란 사용자와 시스템 사이에 일어날 수 있는 상호작용으로, 사용자의 관점에서 요구사항들을 분석할 수 있다. 이는 시스템의 내용을 사용자가 쉽게 이해할 수 있다는 장점이 있다[2]. Use Case 모델을 분석하여 객체들간의 순차적인 메시지 흐름을 도식화하면 상호작용 다이어그램이 만들어진다[3].

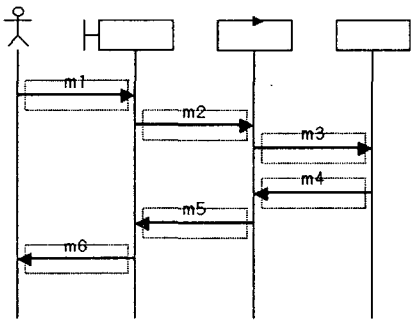
테스트 단계에서, 대규모 프로그램에서 발생할 수 있는 모든 오류를 검색한다는 것은, 한정된 개발시간과 비용으로는 매우 어려운 일이다. 만일 비용을 이유로 테스팅 작업을 소홀히 한다면 그 프로그램의 신뢰성은 낮아지게 된다. 이에 대한 대책으로, 빈번히 발생하는 테스트 데이터들을 뽑아내어 우선적으로 작업을 수행한다면, 어느 정도 해결이 가능하다. 하지만 어떻게 테스트 데이터들을 뽑아내는가는 어려운 문제다. 우리는 Use Case 툴에서 찾아낸 컴포넌트들을 기반으로, 각각의 유한결정 오토마타(deterministic finite automata)를 반자동(semi-automatic)으로 생성하고, 또한 생성된 유한결정 오토마타를 이용해 테스트 플랜 매트릭스(test plan matrix)[10,11]를 구성하여 우선 순위의 테스트 데

이터 선택(test data selection)작업을 수행하고자 한다. 이 논문에서는, 테스팅 단계에서 필요한 설계 컴포넌트들을 5가지로 정의하고, 상호작용 다이어그램에서 각각의 컴포넌트들을 설계하고 생성해 내는 과정을 변환 알고리즘[11]을 적용한 툴에 의해 구현하고자 한다. 이를 통해 유효성과 신뢰성을 갖춘, 재사용 가능한 테스트 데이터들을 뽑아내고자 한다. 본 논문의 구성은 다음과 같이 구성된다. 제2장에서 설계 컴포넌트 단위들의 정의를 언급하고, 제3장에서는 상호작용 다이어그램에서 action matrix로 변환하는 변환 알고리즘을 제시한다. 제4장에서는 구현중인 Use Case툴을 이용해 간단한 예를 적용하고, 제5장에서는 결론을 논하기로 한다.

2. 설계 컴포넌트 정의

2.1 메소드 컴포넌트(Method Component Unit: MCU.)

· 정의 : 메시지에 응답하는 하나의 객체에 의해 실행되는 메소드.



(그림 1) 메소드 컴포넌트의 예

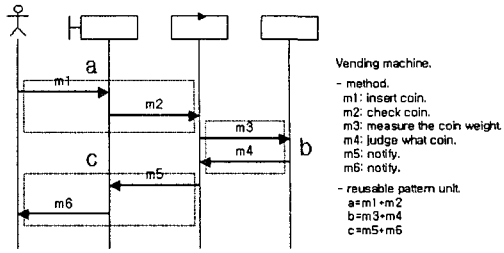
(그림 1)의 단순한 순차 다이어그램을 살펴보자. m1~m6의 구별되는 메시지(function, operation, method)를 메소드 컴포넌트 단위로 보여주고 있다. 메소드 컴포넌트란 하나의 객체에 발생되는 메시지 단위를 의미한다. 이 방법은 메시지 발생만 체크하면 컴포넌트로 구분할 수 있기 때문에 분류는 쉽지만 복잡한 시스템의 경우, 너무 많은 메소드 컴포넌트 발생으로 복잡한 유한결정 오토마타가 생성되어 그로인한 테스팅 수행횟수가 증가되므로, 효율성 측면에서는 문제가 있다.

2.2 재사용 패턴 컴포넌트 (Reusable pattern Component Unit: RCU.)

· 정의 : 특정한 객체에 의해 순차적으로 수행되는 메소드들.

(그림 2)에서는 재사용 패턴 컴포넌트에 기반한 메시지들의 그룹화를 수행하는 과정을, 간단한 자판기의 동작을 예를 들어 설명하고 있다.

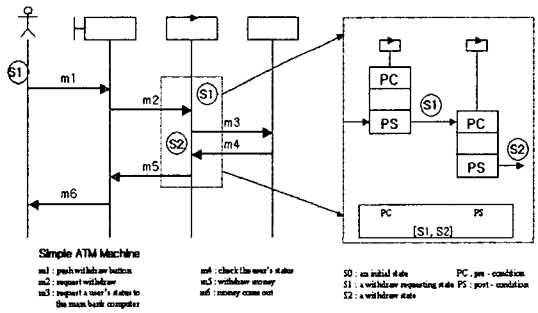
상호작용 다이어그램내의 객체 중, 입력되는 메시지에 대해, 시스템 동작흐름에 영향을 미치는 객체인 제어객체(control object)는 기준요소가 될 수 있다. 여기서 재사용 패턴 컴포넌트란 메시지가 각각의 객체를 통해 발생하여 제어객체를 만날 때까지의 메시지들을 묶는 방법이다. 이 방법은 메시지 발생에 따른 행위들을 일정한 패턴으로 정의하여 구분하기 때문에, 중복된 메시지들을 묶어(clustering) 재사용에 사용함으로써 시간적, 경제적 비용을 줄일 수 있다는 장점이 있다.



(그림 2) 재사용패턴 컴포넌트의 예

2.3 상태 컴포넌트(State Component Unit: SCU.)

· 정의 : 이벤트 상태 모델[4]의 기반에서, 연속적인 상태들간의 간격동안 순차적으로 실행되는 메소드들.



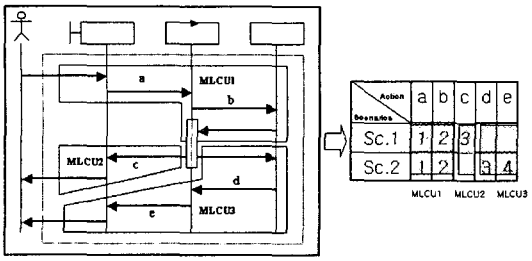
(그림 3) 상태 컴포넌트의 예

(그림 3)에서는 이벤트 상태 모델[4]을 이용한 간단한 현금자동 인출기 예를 보여주고 있다. 이벤트 상태 모델이란 이벤트에 대해, 제어객체가 조건에 따라 결정한 상태변화를 나타낸 것이다. 다시 말하면, 상태 컴포넌트에서는 메시지 발생에 따른 응답과 상태변화를 컴포넌트 단위로 정의하여 구별한다. 이렇게 생성된 상태 컴포넌트는 코드 생성에 주요한 역할을 수행한다.

2.4 최대 선형화 컴포넌트 (Maximal Linear Component Unit: MLCU.)

· 정의 : 액터와 객체사이에서 수행되는 연속적인 메소드들.

(그림 4)에서는 제어객체에 그려진 사각형 bar(object

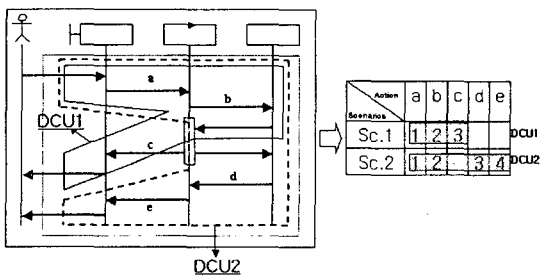


(그림 4) 최대 선형화 컴포넌트의 예

choice)가 메시지들을 받아 판단 · 결정(decision)하여 메시지 흐름을 변경시키는 과정을 보이고 있다. 만일, 메시지가 객체선택 bar를 만나지 않는다면, 메시지의 순차는 시스템 내에서 변하지 않고 일정한 방향으로만 진행하게 된다. 즉, 메시지들이 사각형 bar를 만날 때까지의 흐름단위를 최대 선형화 컴포넌트로 정의하여 데이터의 중복성을 줄이고 재사용할 수 있는 단위로 구성하는 방법이다.

2.5 대화 컴포넌트(Dialogue Component Unit: DCU.)

· 정의 : 액터로부터의 요청(request)과 그 액터가 받는 응답(response)사이에서 실행되는 순차 메소드들.(즉, 사용자가 시스템에 요청한 입력에 대한 응답)



(그림 5) 대화 컴포넌트의 예

대화 컴포넌트란 외부의 사용자가, 내부의 시스템에 요구한 입력내용에 대한 응답을 받을 때까지의 연속된 메시지 흐름을 시나리오화한 것이다. 이 방법은 사용자 승인 테스팅(User Accept Testing)에 적용할 수 있다.

3. 변환 알고리즘(Conversion Algorithm)

Carlson/Hurlburt[4][5]는 Adaptive Use Case의 개념을 제시하고 Use Case에 대한 시나리오들을 action matrix로 표현하는 방법을 제시하였지만, 상호작용 다이어그램에서 action matrix를 발생시키는 메커니즘은 언급하지 않았다. 우리는 그의 접근방법과 유한결정 모델의 Meal[6]과 Moore[7]의 방법을 기반으로, Musa[8][9]의 방법을 결합하여 개선된 action matrix와 Use Case dialog map를 적용하여 우선 순위 테스트 플랜 경로를 제시하였다[10]. 그리고 상호작용 다이어그램에서 개선된 action matrix의 요소들을 찾아내는 알고리즘을 개발하였다[11].

- 알고리즘 전제조건

1. 알고리즘의 입력은 단일제어객체 또는 다중제어객체를 가진 상호작용 다이어그램으로, 출력은 action matrix로 구성된 설계 컴포넌트 단위들의 모음으로 한다.
2. 더 이상 외부 메시지들이 발생하지 않으면, 프로세스는 완료한다.
3. 사용자가 최초 외부 메시지를 시스템에 보낼 때, 사용자의 상태를 S0로 초기화한다.
4. 메시지들이 시스템 안에서 내부메시지로 불려질 경우, 메시지들은 events, methods, actions 로 명명할 수 있다.
5. 시스템 내부의 상태변화 및 경로선택 결정은 제어객체에 의해서만 가능하다.
6. 내부 메시지들이 제어객체에 의해 경로가 선택되어 결정되는 동안 메시지들을 action단위로 묶을 수 있다.
7. 메시지가 내부 시스템에서 빠져 나오면 사용자에게 전달되며, 이 메시지를 외부 이벤트라 한다.
8. 제어객체에 의해 경로선택 결정이 발생하면, 깊이 우선검색 알고리즘을 반복적으로 사용하여 설계 컴포넌트들을 찾는다.

- 상호작용 다이어그램으로부터 설계 컴포넌트 단위를 찾아내는 방법

1단계 : 상호작용 다이어그램에서 최초로 발생되는 외부 메시지로부터, 더 이상 메시지가 발생하지 않을 때 까지를 메소드 컴포넌트로 구분한다.

2단계 : 상태변화 범위 안에서, 순차적 메소드들의 묶음인 재사용 패턴 컴포넌트와 상태 컴포넌트를 구분한다.

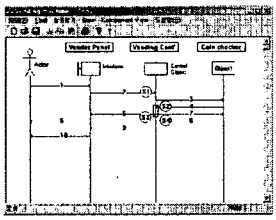
3단계 : 제어객체에 객체선택 bar가 존재하면, 그 곳을 기점으로 전달된 메시지들을 묶어 최대 선형화 컴포넌트로 구분한다.

4단계 : 제어객체에 객체선택 bar가 존재하지 않으면, 시스템과 외부사용자간의 메시지 입력과 응답까지의 메시지 흐름단위를 대화 컴포넌트로 구분한다

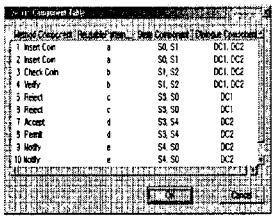
4. 구현

기존의 Use Case 방법론들을 개선하기 위하여, Carlson/Hurlburt/Kim[3][4][11]의 이론들을 적용한 Use

Case 툴을 개발하고 있다. 개발중인 Use Case 툴을 '바이킹'이라 명명하였다. 먼저 정의된 설계 컴포넌트들을 어떻게 추출하고 사용하는지 간단한 자판기 예를 통해 설명하고자 한다. (그림 6)의 '바이킹' 툴에서 상호작용 다이어그램을 설계하면, 자동적으로 우리가 정의한 설계 컴포넌트가 생성된다. 또한 정의된 여러 설계 컴포넌트들중, 개발자나 테스터들의 요구사항에 알맞은 유한결정 오토마타(Deterministic finite automata)들의 생성도 가능하게 된다.



(그림 6) 간단한 자판기의 상호작용 다이어그램



(그림 7) 상호작용 다이어그램으로부터 설계 컴포넌트 구성

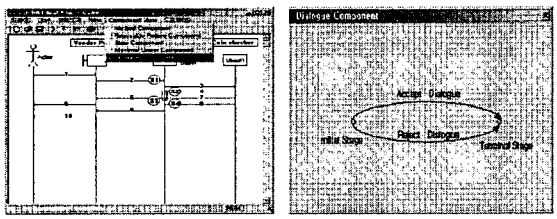
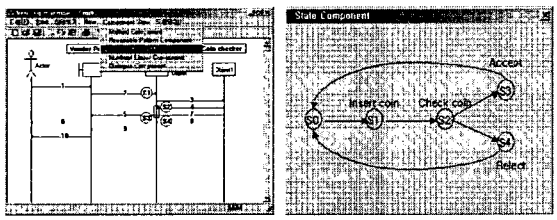


그림 8. 대화 컴포넌트의 유한결정 오토마타



(그림 9) 상태 컴포넌트의 유한결정 오토마타

이 '바이킹' 툴의 장점은 묶음이 큰 유한결정 오토마타를 먼저 테스팅하여 만일 문제경로가 발생되었을때, 그 경로보다 조금 더 자세한 유한결정 오토마타를 선택하여 테스팅 작업과 오류수정이 가능하므로, 복잡성과 비용을 줄일 수 있다는 점이다[10][11].

5. 결과

우리는 기존의 Use Case방법론을 개선하면서, 컴포넌트 기반 방법론으로 전이하려는 목적을 가지고 있다. 현재는 simple과 recursive 유한결정 오토마타들에 대해 비확실성이 있는 등 개선할 여지가 많이 있지만, 이 '바이킹' 툴에 더 많은 기능을 첨가하고 수정작업을 진행하면서 계속 개발할 예정이며, 결국에는 내장형 시스템을 개발하는데 유용하게 사용될 것이라 믿는다. 또한 이들의 유한결정 오토마타에 Kim[10][11]의 테스트 플랜 매트릭스 기법을 적용하여, 우선순위를 가진 테스트 데이터 선택과 테스트 경로의 재사용이 가능한 기능이 추가된다면 더욱 유용한 툴이 될 것이다. 향후 자동으로 완벽한 코드를 발생하는 프로그램을 완성하고자 한다.

6. 참고문헌

[1] Desmond Francis D'Souza and Alan Cameron Wills,"Objects, Components, and FrameWorks with UML : The Catalysis Approach", Addison-Wesley Object Technology Series Oct, 1998.

[2] Firesmith, D."Use cases: The Pros and Cons," ROAD, Vol.2. No2, pp2-6, 1995.

[3] C. K. Low and R. Ronnquist,"Formalism of Interaction Diagram", 3rd Asia-Pacific Software Engineering Conference, pp.318-327, IEEE Computer Society Press, 1996.

[4] Carlson, C. R."Object-Oriented Modeling and Design", Lecture Notes, Computer Science Department, Illinois Institute of Technology, 1999.

[5] Hurlburt, R."Managing Domain Architecture Evolution through Adaptive Use Case and Business Rule Models", PH. D. Thesis, Illinois Institute of Technology, 1998.

[6] Mealy, G. H. "A Method for Synthesizing Sequential Circuits", Bell System Technical Journal, Vol.34, pp.1045-1079, 1956.

[7] Moore, E. F. "Gedanken Experiments on Sequential Machines", In Automata Studies. Annals of Mathematical Studies #34. NJ, 1956.

[8] Musa,J.D. "The Operation Profiles in Software Reliability Engineering: An Overview", AT&T Bell Laboratories Murray Hill, NJ, pp.140-154, 1992.

[9] Musa, J. D."Operational Profile in Software Reliability Engineering: An Overview", Bell Laboratories, pp.14-32, 1993.

[10] Kim, Y. C. and Carlson, C. R. "Scenario Based Integration Testing for Object-Oriented Software Development", Proceeding on The Eighth Asian Test Symposium(ATS'99), Nov. 16-19, 1999.

[11] Kim, Y. C. "A Use Case Approach to Test Plan Generation Design", Ph. D. Thesis Illinois Institute of Technology, May 2000.