제28회 한국정보처리학회 추계학술발표대회 논문집 제14권 제2호 (2007. 11)

행위 모델의 변환을 이용한 수직적 분할 시험

서광익O 최은만
동국대학교 컴퓨터공학과
bradseo@dongguk.eduO emchoi@dgu.ac.kr

Vertical Division Testing by Model Transformation of Activity Model

Seo Kwang IkO Choi Eun Man
Dept. of Computer Engineering, Dongguk University

요 약

최근 활발히 진행되고 있는 모델 기반 공학에 관한 연구 중 모델 변환은 소스 모델을 입력 받아 다른 차원의 뷰를 제공하는 타겟 모델을 출력한다. 이러한 모델 변환은 메타모델을 사용하여 동일한 시스템을 서로 다른 이해관계자들의 관점에서 이해할 수 있는 방법을 제공한다. 동일한 시스템이라 하더라도 개발자와 시험자 그리고 사용자들이 보는 주요 관점은 다를 수 있다. 본 논문에서는 시험자의 관점에서 수직적 분할 시험이 가능하도록 입력 모델인 UML의 행위 다이어그램으로부터 출력 모델인 단위 시험을 위한 상태 다이어그램으로의 모델 변환에 대해 연구하고, 생성된 상태 다이어그램을 통해 시험 사례를 작성한다.

1. 서 론

최근 MDE(Model-Driven Engineering) 방법 중 모델 변환은 소프트웨어의 설계로부터 구현하기 까지 중요한 요소로 부각되고 있다. 모델은 시스템의 특정 부분을 표현하는 객체들과 이들 간의 관계를 표현한 집합이라 할 수 있다. 이러한 모델을 구성하고 있는 요소와 관계들을 쉽게 인지할 수 있도록 그래픽적인 도형을 이용하여 표현한다. 이렇게 도식화된 모델은 이해 관계자들에 따라 모델의 내용과 의미 또는 표현 방법이 서로 상이할 수 있다. 그 예로 UML로 표현된 객체지향형 모델을 관계형으로 표현하거나 시스템을 리팩토링하기 위해 모델 변환을 하는 등 사용 주체와 목적에 따라 다양하게 응용되면서 사용 범위를 넓히고 있다[1][2].

모델 변환은 대부분 모델로부터 소스 코드를 생성하거나 또는 소스 코드를 생성하기 전에 다양한 각도에서 시스템 설계를 재조명하고 하는 것이 일반적이다. 하지만 만약 설계 단계에서 작성된 모델이 시험 단계에서 사용될 수 있도록 시험에 필요한 정보들을 포함한 모델로 변환되거나 변환된 모델이 시험 사례를 작성하는데 도움이 된다면 시험자의 입장에서는 시스템을 시험하기 전에 개발자가 작성한 모델을 이해할 필요 없이 변환된 모델 만을 이용해서 시험을 수행할 수 있을 것이다.

모델 변환은 크게 모델에서 코드로 전환하는 방법과 모델에서 모델로 전환하는 방법으로 구분할 수 있다[1].

모델에서 코드(Model-to-Code)로 전환하는 방법은 특정 모델을 입력 받아서 XML이나 특정 언어의 소스 코드로 변환하는 것을 말한다. 그리고 모델에서 모델(Model-to-Model)로의 변환은 입력된 모델이 내포하고 있는 모듈간의 방향성이나 관계성 또는 구조들을 분석하여 이를 다른 관점으로 변환하여 타겟 모델을 생성한다. 모델 간 변환을 위한 코드의 복잡성을 줄이면서 재사용성을 높이기 위해 메타 모델을 이용한다. 소스 모델의 메타 모델을 참조하여 타겟 메타 모델을 생성하는데, 메타 모델을 생성하는 과정에서 시험을 위한 데이터를 추가하고 이를 이용해서 시험자가 사용할 수 있는 타겟 모델을 생성하면 시험 모델을 생성할 수 있다.

객체지향기반 개발 방법론을 이용하여 시스템을 개발할 경우 UML을 이용하여 시스템을 설계한다. 그리고 설계의 산출물로 생성된 UML을 이용한 모델 기반 시험 기법들이 다양하게 제안되고 있다. 이러한 모델 기반 시험은 두 가지의 제약이 있다. 첫째로 대부분의 모델 기반 시험들은 시험 사례 추출을 위해 참조한 모델의 수준에 따라 시험의 추상 단계가 결정되지만 서로 다른 시험 단계간의 연계 방법이 결여되어 있다. 예를 들면 시스템의 동적 모델링을 위한 순서 다이어그램 또는 협력 다이어그램은 객체들 사이의 상호협력 과정을 명세하기 때문에 대부분 통합 시험이나 시스템 시험 수준의 시험 사례를 생성한다. 반면 주요 클래스의 특정 시점을 명세한 상태 다이어그램을 이용하여 시험 사례를 추출하는 경우는 단위 시험에 한정된다. 따라서 서로 다른 수준의 시험 간 연계 방법이 필요하다. 둘째는 시험 사례를 추출하기 위해 UML을 참조하지만 추출된 시험 사례 형태는 UML 다이어그램이 아니라 새로운 형태의 다이어그램을 제안한다는 것이다[4]. 이럴 경우 시험자는 UML 뿐 아니라 시험 사례를 이해하기 위해 새로 제안된 다이어그램에 대한 의미를 학습해야 하므로 시험

단계의 업무 부하로 이어질 수 있다. 따라서 본 논문에서는 시스템 모듈 간의 행위를 명세할 수 있는 UML 다이어그램 중 스윔레인(swimlane)이 있는 행위 다이어그램(Activity Diagram)을 이용하여 각 스윔레인에 해당하는 객체를 명세하는 상태 다이어그램으로 변환하는 방법에 대해 연구한다.

2절은 수직적 시험을 위한 행위 다이어그램과 상태 다이어그램의 변환 절차에 대해 알아보고 3절에서는 시험 타겟 모델을 생성하는데 필요한 메타 모델에 대해 설명한다. 그리고 생성된 시험 타겟 모델을 4절에서 보이고 5절에서 결론을 맺는다.

2. 수직적 시험을 모델 변환

그림 1은 일반적인 모델 변환 프레임워크를 보이고 있다[5]. 모델 변환은 입력 모델과 타겟 모델 사이의 메타 모델을 이용한다. 그림 1은 전형적인 모델 변환 프레임워크로 소스 메타 모델과 타겟 메타 모델이 각각 입력 모델과 출력 모델에 상응하여 서로 다른 메타 모델을 전제로 설명하고 있다.

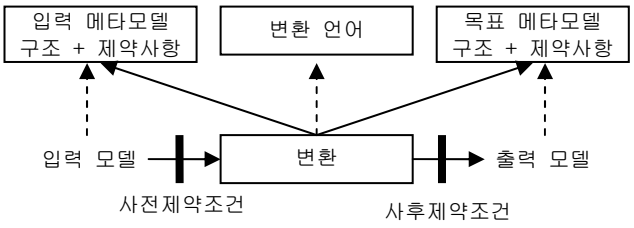

그림 1 모델 변환 패턴

UML 다이어그램 중 행위 다이어그램은 스윔레인이 있어 동작(Action)이나 활동(Activity) 상태의 책임 객체를 나타낼 수 있다. 즉 같은 스윔레(swinlane)인 안에 있는 행위들이 동일한 클래스와 관련이 있다. 따라서 스윔레인을 기준으로 행위 다이어그램을 수직적으로 분할하면 분할된 영역은 하나의 객체에 대한 행위들을 표현하게 된다. 그리고 행위 다이어그램의 행위 결과가 소속된 객체의 상태로 전환이 가능하다면 여러 객체가 협력하는 통합 수준의 행위를 단위 수준의 객체로 분할한 결과가 되어 수직적 분할 및 시험이 가능해진다. 그림 2는 스윔레인을 기준으로 한 행위 다이어그램의 수직적 분할을 보이고 있다. 행위 다이어그램은 추상적 수준이 높은 업무 흐름을 표현하거나 사용자가 시스템을 사용할 때 보게 되는 화면의 흐름뿐 아니라 추상화 단계가 낮은 연산의 알고리즘도 표현할 수 있다. 그림 2는 프로그램의 논리적 흐름을 나타내는 수준으로 표현된 수강 신청 시스템의 일부분이다. 그 중 CourseSection은 학생들의 수강 신청을 처리하는 객체로써 수강 신청을 개설하고 수강 신청 인원에 따라 종료하거나 취소하는 역할을 한다. 그림 2의 행위 다이어그램의 각 행위는 해당 객체의 오퍼레이션 수준으로 명세가 되어 있다. 명세의 추상 수준이 이와 같이 낮은 수준이라면 객체의 상태는 오퍼레이션의 수행 결과에 의해 변경되므로 각 행위를 기점으로 객체의 상태가 변경될 수 있다는 것을 알 수 있다. 따라서 오퍼레이션 수준으로 명세한 행위 다이어그램의 행위가 작동하기 전후의 객체 상태에 대한 정보를 이용하면 객체의 상태 다이어그램을 생성할 수 있다. CourseSection 객체에 소속된 오퍼레이션의 수행 결과에 따른 상태 정보(A: planned, B: open, C: EnoughStudents, D: NotEnoughStudents, E: closed, F: cancel)가 추가되어 있다. 이러한 상태 정보가 메타 모델에 추가되고 상태 다이어그램을 생성할 때 사용될 수 있다.

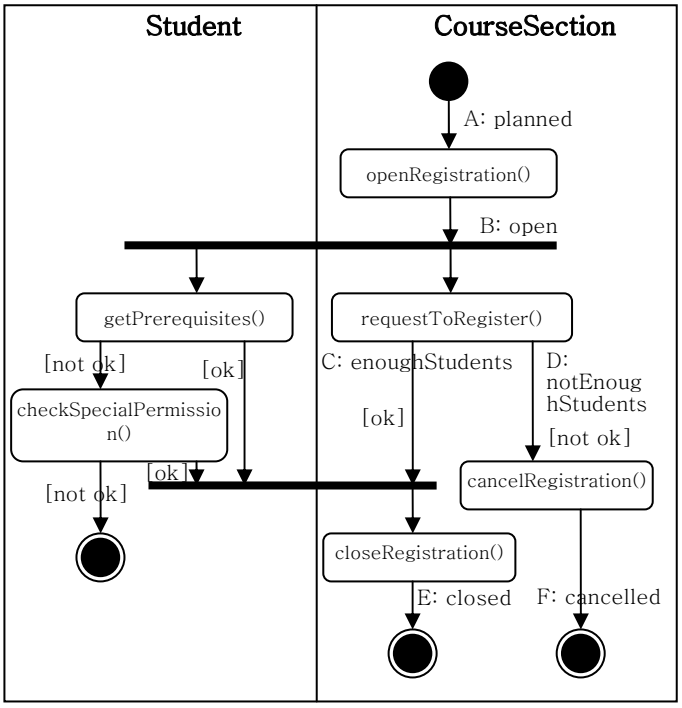

그림 2 행위 다이어그램의 수직적 분할

본 논문에서는 UML의 행위 다이어그램을 입력 모델로 사용하여 행위 모델의 메타 모델을 설계하고 단위 시험을 위한 상태 다이어그램을 생성한다.

3. 메타 모델과 모델 변환

모델 변환은 그림 1을 통해 설명한 바와 같이 입력 모델에 대한 입력 메타모델이 출력 모델에 대한 목표 메타모델로 변환하는 것을 말한다. 본 논문에서는 입력 모델로 행위 다이어그램을 사용하고 목표 모델로는 상태 다이어그램을 생성한다. 그림 3은 행위 다이어그램의 메타모델이다. 그림 2의

행위 다이어그램에 추가된 상태 정보는 모든 행위가 실행된 후에 적어도 하나의 상태를 가질 수 있다. 이러한 관계를 그림 3의 ActivityNode 객체와 State 객체가 실현하고 있다.

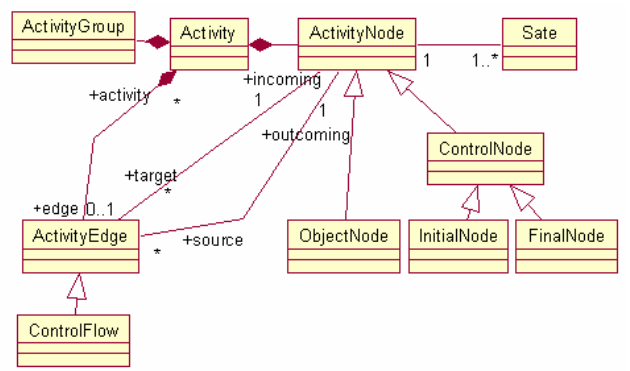

그림 3. 행위 다이어그램의 메타모델

그림 4는 간단한 상태 다이어그램의 상태와 변환을 표현하기 위해 제안한 기본 상태 다이어그램 메타모델이다[6]. 기본 상태 다이어그램 메타모델을 수정하여 시간적 제약사항과 같은 부수적인 정보를 추가할 수도 있다.

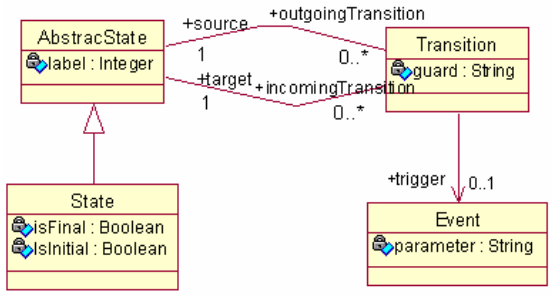

그림 4. 상태 다이어그램 메타모델

그림 4의 상태 다이어그램의 메타모델에 있는 State 객체는 그림 3의 행위 다이어그램의 메타모델을 생성할 때 참조한다. 또한 행위 다이어그램의 행위는 상태 다이어그램의 상태 변환을 일으키는 트리거(trigger)나 이벤트(event)의 역할을 한다.

모델 변환은 메타모델을 구성하고 있는 요소들 간의 관계를 파악하여 모델 사이의 변환을 수행한다. 모델 변환을 위해 이클립스 모델링 프레임워크(Eclipse Modeling Framework: EMF)[7] 기반의 모델 변환 엔진인 Tefkat[8]을 사용할 수 있다. Tefkat은 소스 모델과 소스 메타모델 그리고 타겟 메타 모델과 규칙(rules)을 입력 받아 타겟 모델을 생성한다. 소스 모델에서 타겟 모델로 변환하기 위한 규칙을 실행하기 위해 소스 메타모델과 타겟 메타모델이 필요하다. 그림 5는 행위 다이어그램의 모델과 메타모델 그리고 상태 메타모델과 변환 규칙을 입력 받아 타겟 모델을 생성하는 Tefkat 변환 엔진을 설명하고 있다.

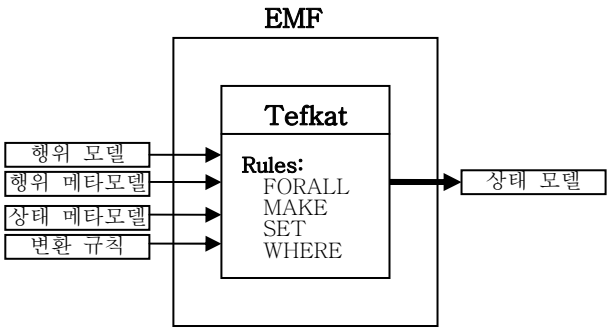

그림 5. 모델 변환 명세

4. 시험 모델

행위 다이어그램을 참조하여 통합 시험이나 시스템 시험을 수행하다가 오류를 발견하면 오류를 수정하기 위해 단위 시험을 해야 한다. 2 절에서 이미 언급한 바와 같이 행위 다이어그램은 스윔레인(swinlane)을 제공하고 있어서 각 행위에 대한 책인 주체를 밝히고 있다. 따라서 그림 2의 행위 다이어그램을 이용하여 모델 기반 통합 시험을 한 후 오류가 발견되면 객체 하나하나를 따로 분리하여 시험하여 하는데, 그림 6과 같이 스윔레인을 기준으로 단위 시험 수준의 객체의 변화를 보여주는 상태 다이어그램을 생성한다면 이는 통합 시험 수준에서 단위 시험 수준으로 전환이 용이하게 된다.

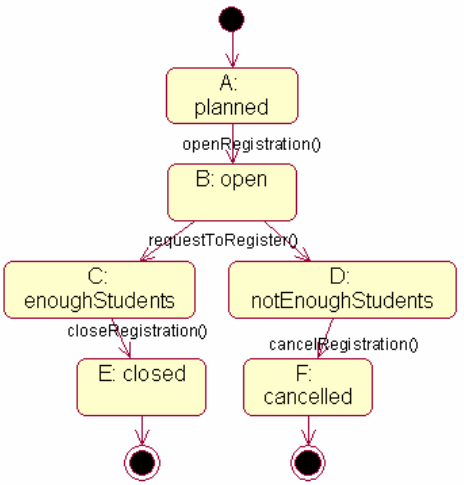

그림 6. 단위 시험을 위한 CourseSection 객체의 상태 다이어그램

그림 6을 통해 CourseSection 객체를 시험하기 위한 경로는 두 가지(A-B-C-E, A-B-D-F)가 있음을 알 수 있다. 그리고 각 상태가 가져야 하는

상태 정보를 표 1의 원시 코드에서 파악하여 각 상태에 해당하는 시험 사례를 작성할 수 있다.

표 1. CourseSection 원시 코드

```
public class CourseSection
{
        ...
    //A: planned
        private boolean open = false;
        private boolean closedeOrCancelled = false;
        private boolean NotEnoughStudents = false;
        private boolean EnoughStudents = false;

        public CourseSection(Course course)
        {
                this.course = course;
                RegistrationList = new LinkedList();
        }

        public void openRegistration()
        {
                if(!closedOrCancelled)
                {
                        open = true; //B: open
                }
        }
public void requestToRegister(Student student)
        {
        if (open)
        {
                ... 중략 ...

        if (registrationList.size() >= course.getMaximum())
                {       //C: enoughStudents, open의 substate
                    enoughStudents = true;
                    closeRegistration();
                }
        else //registrationList.size() < course.getMinimun()
                {       //D: notEnoughStudents, open의 substate
                    notEnoughStudents = true;
                    unregisterStudents();
                    closeRegistration();
                }
        }
public void closeRegistration()
        {       //E: closed
                open = false;
                closedOrCancelled = true;
        }
public void calcel()
        {       //F: cancelled
                open = false;
                closedOrCancelled = true;
                unregisterStudents();
        }
//Private method to remove all registrations
private void unregistraterStudents()
        {
                ... 중략 ...
        }
}
```

표 2는 그림 6에서 생성한 두 개의 경로와 각 경로에 해당하는 상태를 결정하는 멤버 변수의 값들을 이용하여 단위 시험 사례를 생성하였다.

표 2. 단위 시험 사례

경로	멤버변수	상태 A	B	C	D	E	F
1	open	false	true	true	–	false	–
	closedOrCnacelled	false	false	false	–	true	–
	enoughStudents	false	false	true	–	false	–
	notEnoughStudents	false	false	false	–	false	–
2	open	false	true	–	true	–	false
	closedOrCnacelled	false	false	–	false	–	true
	enoughStudents	false	false	–	false	–	false
	notEnoughStudents	false	false	–	true	–	false

5. 결론

모델 기반의 개발에 기초를 둔 모델 기반의 시험이 최근 다양하게 제안되고 있다. 이러한 모델 기반 시험의 장점은 이미 정의된 산출물을 이용한다는 점에서 자동화를 쉽게 할 수 있고 시간과 자원을 절약할 수 있을 뿐 아니라 시험 단계에서 수행할 작업들을 개발 초기로 앞당길 수 있다는 것이다. 본 논문에서는 모델 변환을 통해 개발 초기에 기능 수준으로 작성된 행위 다이어그램을 사용하여 단위 시험을 위한 상태 모델을 생성하고 시험 사례를 생성했다. 이러한 작업은 행위 모델로 작성된 기능 시험을 수행하다가 실패한 시험 사례에 대해 더욱 자세한 단위 시험을 할 수 있는 방법을 제시하고 있다.

향후 연구 과제로 출력 모델인 상태 다이어그램을 생성하는 과정에서 추가해야 하는 시험 데이터와 실행 가능한 소스 수준의 시험 사례를 생성해 본다. 또한 스윔레인에 해당하는 책임 객체가 생략될 때 발생하는 문제를 보완하기 위한 테스트 스텁(stub)에 관해 연구한다.

6. 참고 문헌

[1] Specifying Model Transformations at the Metamodel Level, S. R. Judson, R. B. France and D. L. Carver,

[2] Model Transformation with the IBM Model Transformation Framework

[3] Classification of Model Transformation Approaches, K. Czarnecki and S Helsen, OOPSLA 03' Workshop on Generative Techniques in the Context of Model-Driven Architecture,

[4] Rigorous Testing by Merging Structural and Behavioral UML Representations, O. Phliskalns, A. Andrews, S. Ghosh and R. France.

[5] Model Transformation Testing Challenges, B. Baudry, T. Dinh-Trong, JM. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, Y. L. Traon.

[6] Metamodel-based Test Generation for Model Transformatins: an Algorithm and a Tool, E. Brottier, F Fleurey, J. Steel, Be. Baudry, Y. L. Traon, 17th International Symposium on Software Reliability Engineering(ISSRE'06).

[7] EMF: http://www.eclipse.org/emf

[8] Tefkat: http://tefkat.sourceforge.net