신기술 해설

모델 변환 기법과 변환 중심 테스트

목 차

최 은 만
(동국대학교)

1. 서 론

모델은 소프트웨어 개발에서 점차로 중요해지고 있다. 모델 중심 아키텍처(Model Driven Architecture)가 관심을 끌면서 연구나 도구들이 모델을 소프트웨어 엔지니어링의 중요한 결과물로 강조하고 있다. 모델은 개발자나 고객 등 소프트웨어 관련자들이 시스템에 대한 관심이나 질문, 변경 등을 하려 할 때 효과적으로 만드는 시스템의 추상적 표현이다.

모델 변환(model transformation)[1]은 다음과 같은 여러 가지 목적으로 한 모델에서 다른 모델로 바꾸는 것을 말한다. 먼저 제일 흔한 적용은 상위 모델에서 하위 모델로 결국은 코드 단계까지 전환하는 것이다. 추상 수준이 같은 모델들 사이에 매핑 또는 동기화하기 위하여 또는 모델을 개선시키고 발전시키기 위한 모델 리팩토링을 목적으로 할 수도 있다. 역으로 코드나 낮은 수준의 추상 모델로부터 높은 수준의 모델을 추려내는 작업에도 필요하다.

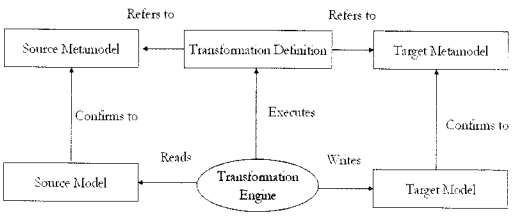
모델 변환은 프로그램 변환이나 메타 프로그래밍보다는 상대적으로 초기 단계의 연구이나 점점 관심이 높아가는 이유는 모델 변환이 모델 중심의 소프트웨어에서 핵심 역할을 하고 있으며 소프트웨어가 발전, 진화되면서 변경이 필수적으로 따르며 이러한 변화가 여러 모델에 반영되어 동기화 되지 않는다면 문제가 심각하며, 모델의 변환 과정을 체계화시키면 모델의 변화 또는 리팩토링 과정을 효과적으로 추적할 수 있기 때문이다. 추가적으로 대규모의 소프트웨어 시스템 제작에는 여러 가지 다른 관점의 모델링이 필요하며 이들 사이에 일관성이나 이해를 위하여 통일시킬 필요가 있는 경우도 있다.

2. 모델 변환 접근 방법

모델 변환은 모델에 가해지는 것이므로 모델이 무엇인지 명확히 할 필요가 있다. 모델은 프로그램 코드에 나타난 의미들을 추상화 한 것이다. 모델은 프로그램보다 그 형태가 다양하므로 프로그램 변환에 사용하는 접근 방법보다 훨씬 다양하다. 대부분의 모델이 프로그램보다는 비주얼 행태를 가지고 있으나 인터페이스 명세, 컴포넌트 명세, 사용 사례 명세, 테스트 모델 등은 비주얼 타입이 아닌 경우도 있다. UML과 같은

비주얼 타입의 모델은 그래프 중심 변환 방법이 많이 사용된다. 이 절에서는 다양한 형태의 모델을 변환하기 위한 접근 방법에 대하여 소개한다.

모델 변환 방법이 무엇이든 기본적인 모델 변환 개념은 (그림 1)과 같다. 소스 모델과 타깃 모델에 대한 메타 모델, 즉 모델링 표현의 추상 구문을 정의하고 메타 모델 사이의 변환을 미리 정의한다. 구체적인 한 모델의 변환은 이러한 미리 정의된 메타 모델의 변환의 인스턴스가 된다.



(그림 1) 모델 변환의 기본 개념

2.1 직접 조작

직접 조작이란 내부 모델 표현과 이를 조작하는 API 들을 제시하는 것으로 객체지향 프레임워크로 구현된다. 따라서 변환을 위하여 추상 클래스 등을 제공하지만 실제 사용자가 이를 이용하여 변환 규칙을 구현하여야 한다.

2.2 구조 중심

구조 중심 변환은 두 가지 단계로 구성된다. 첫째는 타겟 모델의 계층적 구조를 생성하는 단계이며 두 번째 단계는 타겟에 속성과 레퍼런스를 추가하는 것이다. 전체 프레임워크가 스케줄링과 적용 방법을 결정하고 사용자는 변환 규칙만 제공하면 된다.

OptimalJ[2]가 제공하는 모델 변환 프레임워크가 여기에 해당되는데 프레임워크는 Java로 구현되어 점증적인 copier를 제공하며 사용자는 이를 서브클래스로 상속하여 자신의 변환 규칙을 정의한다.

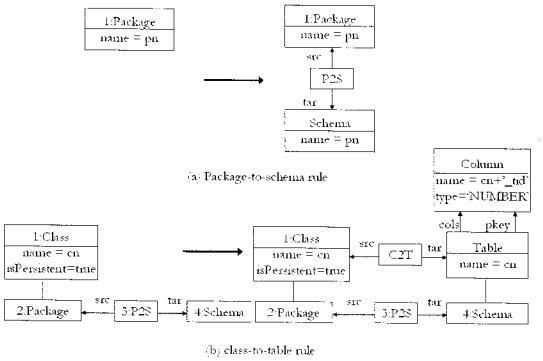
2.3 오퍼레이션 중심

이 방법은 직접 조작과 유사하나 모델변환을 위한 전용 기능을 제공한다는 면에서 다르다. 예를 들면 정형화된 메타 모델에 expression computation을 위한 기능을 제공하여 확장하는 방법이 있는데 OCL과 같은 질의어를 명령어 구조로 확장하는 것이 구체적인 사례가 될 수 있다. 대표적인 사례가 QVT 오퍼레이션 언어이다. 다음 사례가 UML 모델을 RBBMS 모델로 변환하기 위하여 QVT 오퍼레이션 언어로 작성한 것이다. 모델의 매핑은 모델 요소에 대한 오퍼레이션으로 정의된다. 예를 들어 packageToSchema는 Schema를 리턴 타입으로 하는 Package의 오퍼레이션이다. 매핑 구조 안에는 리턴 객체의 속성을 내장한다. 자세한 QVT 오퍼레이션 명세는 [3]에 소개되어 있다.

```
transformation umlRdbms{
   in uml: SimpleUML,
   out rdbms: simpleRBMS
};
main() {
   uml.objectsOfType(Package->map
packageToSchema();
   uml.objectsOfType(Attribute->map
attributeToSchema();
}
mapping Package::packageToSchema(): Schema {
name:=self.name;
   tbls:=self.elems->map classToTable();
}
mapping Class::classToTable(): Table
   when { self.isPersistent=true; } {
   name:=self.name;
   key:=object Column {
                  name:=self.name+'_tid';
                  type:='NUMBER';
         };
   cols:=key;
}
```

3.4 그래프 중심

이 접근 방법은 그래프 변환의 이론적인 작업에 속한다. 대부분 타입이 있고 속성이 포함된 레이블 그래프를 사용한다. 그래프 변환 규칙은 LHS와 RHS 패턴으로 구성되는데 변환될 모델이 LHS 패턴에 일치하면 RHS 패턴으로 대치하는 것이다. 그림 2의 두 가지 규칙 중 화살표 왼쪽의 패턴이 나오면 오른쪽 패턴으로 바꾸면 모델이 변환된다.



(그림 2) 그래프 중심 변환

3. 모델 변환에 의한 테스트

테스트 작업은 크게 두 가지 단계로 나눈다. 첫째는 클래스 다이어그램이나 순차 다이어그램에서 테스트 모델, 즉 책임 중심 테스트 모델을 세우는 것이다. 이 작업은 주로 추상적 수준에서 이루어지며 플랫폼 독립적이다. 책임 중심 테스트 모델은 테스트 시나리오 형태로 작성되기도 하며 그래프 형태로 나타내기도 한다. 다음 단계는 테스트 모델을 실행 가능한 테스트 케이스들로 변환하는 작업이다. 특히 최근에 이 단계의 작업들을 자동화 하거나 패턴화[4] 하려는 연구나 작업들이 이루어지고 있다.

여기서 자세히 예를 들지는 못했지만 순차 다이어그램을 이용하여 테스트 모델을 세우고 이를 테스트 케이스로 변환하는 방법도 사용할 수 있다. 순차 다이어그램의 특정 라이프 라인으로부터 메소드 호출을 선택하여 이를 구동하는 테스트 케이스를 만들고 결과를 비교하는 방법이다. 여기에서 주의할 점은 후속 라이프 라인에서 구동되는 메소드 호출은 원래의 메소드 호출에 의하여 간접적으로 구동되는 것이다. 따라서 테스트 케이스를 실행하는 동안 추적하고 결과를 비교 체크할 필요가 있다.

또 한 가지 중요한 변환 방법은 애스펙트 중심의 변환이다. 성능을 테스트 한다든지, 특정 부분의 동작, 또는 특정 데이터 값의 변화를 추적하기 위하여 여러 클래스에 분산되어 있는 부분들을 별도의 애스펙트로 정의하고 이를 AOP에 의하여 직조함으로 테스트 작업을 효과적으로 구성할 수 있다. 이러한 아이디어들은 [5]와 같이 보안 테스트나 성능 테스트와 같은 시스템 테스트에 적극 활용되고 있다. 그 이유는 원시코드의 작성이 완성되어 빌드한 후 시스템을 블랙박스 형태로 유지하면서 필요한 체크나 호출 또는 변경을 가하기 위함이다.

4. 결 론

이제까지 모델의 변환은 어떤 형태가 있으며 모델 변환이 테스트 작업에 어떻게 이용될 수 있는지 살펴보았다. 모델 중심 개발이 더욱 활발해지면서 원시코드의 변형이나 리팩토링보다는 설계의 변형이나 리팩토링, 변환이 매우 중요해졌다.

테스트 모델은 크게 설계 모델과 구현을 바탕으로 구축된다. 최근 테스트 병행 개발이 주목을 끌면서 구현 이전, 즉 설계 단계에 미리 테스트 모델을 세우고 이를 검증하는 방법들을 도입하고 있는데 이 연구에서는 설계 모델에서 어떻게 테스트 모델로 변환해 나가느냐를 다루었다. 설계의 어떤 부분이 어떤 책임을 감당하고 있는지 체크하는 테스트 모델을 UML에서 꺼내서 ECL로 기술할 수 있음을 보였고 일반적인 테스트

모델로의 변환 프로세스가 어떻게 될 것인지를 개관적으로 설명하였다.

참고문헌

[1] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM System Journal, Vol. 45, No. 3, pp. 621-645, 2006.

[2] OptimalJ 4.0 User Guide, Compuware, http://www.compuware.com/products/optimalJ, June, 2005.

[3] OMG, MOF QVT Final Adopted Specification, OMG Adopted Specification ptc/05-11-01, 2005.

[4] Gerard Meszaros, xUnit Test Patterns: Refactoring Test Patterns, Addison-Wesley, 2007.

[5] N. Belblida, et. al, "AOP extension for security testing of programs", Proc. of IEEE Canadian Conference on Electrical and Computer Engineering, Ottawa, pp.647-65, 2006.

저자약력

최 은 만

1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과(석사)
1993년 미국 일리노이 공대 전산학과(박사)
1985~1988년 한국표준연구원 연구원
1988~1989년 (주)데이콤 주임연구원
2000년, 2007년 콜로라도 주립대 전산학과 방문교수
1993년~현재 동국대학교 컴퓨터공학과 교수
관심분야 : 객체지향 소프트웨어공학, 소프트웨어 유지보수, 재사용, AOP