객체지향 프레임워크의 구조적 테스트 패턴 추출 방법

김장래 전태웅
고 려 대 학 교 전 산 학 과
{ future, jeon }@selab.korea.ac.kr

The Method of Structural Test Pattern Extraction for Object-Oriented Framework Test

Jangrae Kim Taewoong Jeon
Dept. of Computer Science, Korea University

요 약

객체지향 프레임워크는 다수의 응용 소프트웨어 개발에 반복적으로 재사용되므로 철저한 시험이 요구될 뿐만 아니라 재사용 시 확장된 프레임워크에 대해서도 추가적인 시험이 필요하다. 본 논문에서는 재사용 시 다양한 형태의 객체 구조들로 개조, 확장될 수 있는 프레임워크의 가변부위에 대해 구조적 테스트 패턴들을 프레임워크의 합성 패턴들로부터 조직적으로 추출하는 방법을 기술한다. 여기서 제안된 방법은 가변부위 클래스 구조의 테스트 모델을 정형 명세화하고, 이로부터 시험에 필요한 구조적 테스트 패턴을 추출하는 방법을 제공한다

1. 서론

현재까지 객체지향 소프트웨어의 효과적인 시험 방법들은 많이 연구되어 왔다[6, 7]. 그러나 프레임워크는 기존의 이러한 시험 방법들만으로는 효과적인 시험을 어렵게 하는 여러 가지 문제들을 지니고 있다. 프레임워크를 시험하기 위해서는 개조 (adaptation), 확장 (extension), 구체화 (instantiation) 등의 작업을 거쳐서 실행 가능한 하나의 완전한 시스템으로 만들어야 한다. 그런데 프레임워크가 (재)사용되는 형태, 상황 및 응용 사례들은 매우 다양할 뿐만 아니라 이를 사전에 명시하고 예측하기가 쉽지 않다.

본 논문에서는 개조, 확장된 객체지향 프레임워크에 대한 조직적인 시험을 효율적으로 수행할 수 있도록 정형 명세된 테스트 모델로부터 구조적 테스트 패턴을 추출하는 방법을 기술한다.

2. 프레임워크의 가변부위

객체지향 프레임워크는 고정부위와 가변부위가 클래스 또는 메쏘드 단위로 분리되어 설계된다. 가변부위와 접속된 고정부위에 해당하는 클래스(메쏘드)를 템플릿 클래스(메쏘드)라고 부르고 가변부위에 해당하는 클래스(메쏘드)를 후크 클래스(메쏘드)라고 부른다[2]. Pree 는 객체지향 프레임워크의 객체 컴포넌트들 사이의 합성 패턴들을 1) 템플릿 클래스 객체에 합성되는 후크 클래스 객체의 개수(복수 허용 여부)와, 2) 상호 합성 관계인 템플릿 클래스와 후크 클래스 사이의 상속 관계 여부의 기준에 따라 그림 1 에서와 같이 7 개의 유형들로 추상화하여 분류하였다[2].

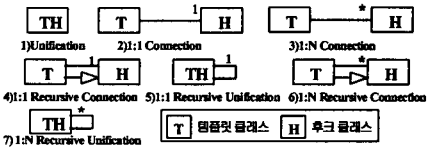


그림 1. 템플릿-후크 클래스 합성 패턴

3. 테스트 패턴

본 논문의 테스트 대상이 되는 CUT(Cluster Under Test)는 프레임워크의 가변부위를 형성하며, 위와 같은 합성 패턴에 따라 다양한 객체 구조로 생성될 수 있다. CUT 모델 중에서 테스트에 필요한 부분만을 추출하거나 테스트에 필요한 정보를 추가해서 테스트 모델을 구성한다[3].

테스트 모델은 테스트 대상이 되는 클래스들의 객체를 생성하고 그 객체들에 대한 테스트 케이스를 구성하기에 충분한 정보를 보유하고 있어야 한다. 테스트 모델의 클래스 클러스터 구조로부터 생성될 수 있는 객체 클러스터의 형태는 매우 다양하며 이러한 객체 클러스터 구조들의 집합은 분류기준(partition criteria)에 의해 분할 영역으로 분류될 수 있다.

오퍼레이션을 테스트하기 위해 Stocks 와 Carrington 이 제안한 테스트 데이터 분류 방법(TTF:Test Template Framework)[4]은 분류기준(heuristic)을 이용하여 오퍼레이션의 입력 데이터 영역을 세분화된 분할 영역으로 분류하며, 이러한 분류는 특정 입력 데이터가 분할 영역을 대표하는 데이터가 될 때까지 반복된다.

본 논문에서는 테스트 데이터 분류 방법을 프레임워크의 가변부위 테스트에 적용하였다.

프레임워크 가변부위의 테스트를 위해서는 테스트 모델의 클래스 클러스터 구조로부터 생성 가능한 모든 객체 클러스터 구조들의 영역을 분류기준을 적용하여 세분화된 분할 영역으로 분류하며, 이때 분류기준은 객체들간의 연결 구조와 연결된 객체의 타입, 수(복수 허용 여부)가 된다. 이러한 분류는 객체 구조의 확장 유형이 상위 단계에서 나타났던 형태와 일치할 때까지 즉, 가능한 모든 형태의 확장이 표현될 때까지 반복된다. 더 이상 세분되지 않는 분할 영역들을 테스트 패턴으로 정의한다. 테스트 모델로부터 추출된 모든 테스트 패턴들은 해당 테스트 모델로부터 생성 가능한 모든 객체 클러스터 구조를 세분화한 결과이며, 특정 테스트 패턴으로부터 도출된 임의의 객체 클러스터 구조는 해당 테스트 패턴을 대표할 수 있다. 그림 2 는 테스트 모델에 분류기준을 적용하여 테스트 패턴을 추출하는 과정을 표현한다.

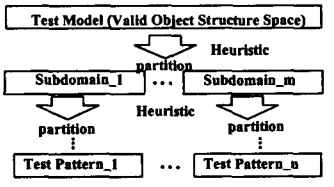
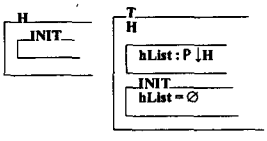


그림 2. 테스트 패턴을 추출하기 위한 객체 클러스터 구조의 분류

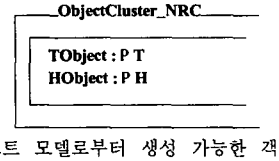
위의 분류과정은 UML[5]과 Object-Z[8]를 이용하여 엄격히 명세화 할 수 있다.

예를 들어, 그림 1 의 1:N *Recursive Connection* (NRC) 합성 패턴을 명세하면 다음과 같다.



1:N Recursive Connection 합성 패턴은 템플릿 클래스 객체가 그 객체에 1 대 N 으로 연결된 후크 클래스의 서브클래스 인스턴스들이다. 템플릿 클래스 객체에 연결된 복수의 후크 클래스 객체들이 다시 각각 템플릿 클래스 객체로서 또 다른 복수의 후크 클래스 객체들에 연결되는 과정을 반복함으로써 트리 구조를 형성할 수 있다.

이러한 테스트 모델로부터 생성 가능한 객체 클러스터의 스키마는 다음과 같고,



테스트 모델로부터 생성 가능한 객체 구조(VOS: Valid Object Structure)를 다음과 같이 정의할 수 있다.

$VOS \cong [\ ObjectCluster\_NRC$

$\quad | \#HObject > 0 \Rightarrow \#TObject > 0\ ]$

테스트 패턴을 추출하기 위한 분할의 첫번째 단계는 VOS 영역을 분할 하며 이때 사용되는 분할 기준으로는 T 클래스의 H 클래스에 대한 인스턴스 변수 hList 의 cardinality 가 0 또는 그 이상으로 선언되어 있으므로 hList 에 연결된 객체의 개수에 기반한 분할(cardinality based partitioning)을 적용할 수 있다.

[HEURISTIC]

$|$ cardinality : *HEURISTIC*

상위 영역을 하위 분할 영역으로 분류하는 분할 함수인 TTH(Test Template Hierarchy)는 계층적인 VOS 분할 구조에서 상위 영역과 분할기준의 쌍을 하위 영역들의 집합으로 대응시키는 함수이다.

$|\ TTH : \mathbb{P}\ VOS \times HEURISTIC \rightarrow \mathbb{P}\ (\mathbb{P}\ VOS)$

따라서 VOS 를 분류한 첫번째 단계의 하위 영역들 스키마와 TTH 함수는 다음과 같다.($TObject_1$: 첫번째 단계의 분할에서 생성된 T 타입의 객체)

$TT_1 \cong [VOS \mid \#\{\ TObject_1.hList\ \} = 0\ ]$

$TT_2 \cong [VOS \mid \#\{\ TObject_1.hList\ \} = 1\ ]$

$TT_3 \cong [VOS \mid \#\{\ TObject_1.hList\ \} > 1\ ]$

$TTH_{NRC}\ (VOS, cardinality) = \{TT_1, TT_2, TT_3\}$

이것을 객체 다이어그램으로 표현다면 그림 3 과 같다.

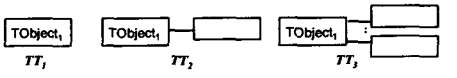


그림 3. NRC VOS 의 cardinality 에 의한 분할 객체 다이어그램

TTH_{NRC} 에 의해 VOS 로부터 세분화된 분할영역 TT_1, TT_2, TT_3 각각의 임의의 인스턴스는 서로 중복되지 않으며 모든 분할 영역들로부터 생성된 인스턴스들의 집합은 다시 VOS 와 일치한다.

$\forall vos : PVOS \bullet (\forall T1, T2 : TTH_{NRC}\ (vos, cardinality) \mid T1 \neq T2 \bullet T1 \cap T2 = \emptyset)$

$\forall vos : PVOS \bullet (\forall T : TTH_{NRC}\ (vos, cardinality) \bullet \cup T = vos)$

위의 첫번째 단계에서 도출된 세 개의 영역들은 다시 다음 단계에서 세분화될 수 있다. 두 번째 단계에서는 참조변수 hList 에 할당된 참조대상 객체의 타입에 의한 타입 기반 분할기준(type based heuristic)을 적용하여 즉, *hList* 에 T 클래스 타입의 객체가 몇 개 할당되어 있는가에 따라 분할하며 그 결과는 다음과 같다.

| type_based : HEURISTIC
$TTH_{NRC}(TT_1, type\_based) = \{TT_1\}$

$TTH_{NRC}(TT_2, type\_based) = \{TT_{2.1}, TT_{2.2}\}$
$TT_{2.1} \cong [\, TT_2 \mid \#\{t : TObject \mid t \in TObject_1.hList \wedge t \neq TObject_1\} = 1 \,]$
$TT_{2.2} \cong [\, TT_2 \mid \#\{t : TObject \mid t \in TObject_1.hList \wedge t \neq TObject_1\} = 0 \,]$

$TTH_{NRC}(TT_3, type\_based) = \{TT_{3.1}, TT_{3.2}, TT_{3.3}\}$
$TT_{3.1} \cong [\, TT_3 \mid \#\{t : TObject \mid t \in TObject_1.hList \wedge t \neq TObject_1\} > 1 \,]$
$TT_{3.2} \cong [\, TT_3 \mid \#\{t : TObject \mid t \in TObject_1.hList \wedge t \neq TObject_1\} = 1 \,]$
$TT_{3.3} \cong [\, TT_3 \mid \#\{t : TObject \mid t \in TObject_1.hList \wedge t \neq TObject_1\} = 0 \,]$

위에서 TT_1 은 hList 가 공집합으로 어떤 타입의 객체도 할당될 수 없으므로 더 이상 분할되지 않는 영역임을 알 수 있다. 이와 같이 더 이상 분할되지 않거나, 영역이 분할될 때 객체 클러스터 구조의 확장 형태가 반복적으로 나타나면 테스트 패턴으로 정의하고 더 이상 분할하지 않는다.

$TT_{3.1}$ 과 $TT_{3.2}$ 는 다음 단계에서 계속 분할되나, $TT_{2.1}$ 은 분할될 때 객체 클러스터 구조 확장 형태가 VOS 에서 하위 영역으로 분할될 때와 같고 $TT_{2.2}$, $TT_{3.3}$ 은 더 이상 분할되지 않으므로 테스트 패턴이다. 지금까지 추출된 테스트 패턴을 Object-Z 명세로 표현하면 다음과 같다.

$TT_1 \cong [ObjectCluster\_NRC \mid \#TObject = 1 \wedge \#HObject = 0]$
$TT_{2.1} \cong [ObjectCluster\_NRC \mid \#TObject = 2 \wedge \#HObject = 0 \wedge (t1, t2 : TObject \mid t1 \neq t2 \wedge t2 \in t1.hList)]$
$TT_{2.2} \cong [ObjectCluster\_NRC \mid \#TObject = 1 \wedge (t : TObject \mid (\exists_1 h : HObject \bullet h \in t.hList))]$
$TT_{3.3} \cong [ObjectCluster\_NRC \mid \#TObject = 1 \wedge \#HObject > 1 \wedge (t : TObject \mid (\forall h : HObject \bullet h \in t.hList))]$

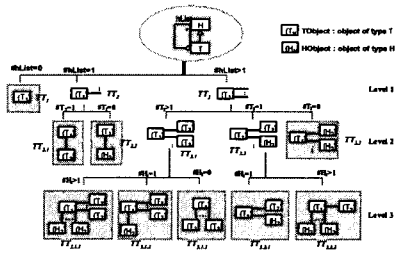


그림 4. 1:N Recursive Connection 테스트 모델의 구조적 테스트 패턴 추출

$TT_{3.1}$ 과 $TT_{3.2}$ 는 계속해서 분할될 수 있으므로 위의 과정을 반복적으로 적용하면 모두 9 개의 테스트 패턴(TT_1, $TT_{2.1}$, $TT_{2.2}$, $TT_{3.3}$, $TT_{3.1.1}$, $TT_{3.1.2}$, $TT_{3.1.3}$, $TT_{3.2.1}$, $TT_{3.2.2}$)을 추출할 수 있다. 각 단계의 분할 과정을 객체 다이어그램으로 표현하면 그림 4 와 같다. 최종적으로 추출된 9 개의 테스트 패턴 역시 모든 테스트 패턴으로부터 생성 가능한 객체 클러스터들의 합집합은 테스트 모델로부터 생성 가능한 객체 클러스터들의 집합과 같으며, 서로 다른 테스트 패턴에서 생성된 인스턴스들은 중복되지 않는다.

각각의 테스트 패턴으로부터 생성된 객체 클러스터 즉, 테스트 패턴 인스턴스는 해당 테스트 패턴을 대표하므로 객체지향 프레임웍의 가변부위를 테스트하기 위한 객체 연결 구조로 사용될 수 있다.

4. 결론

본 논문에서는 객체지향 프레임워크의 가변부위 클래스 구조의 테스트 모델을 정형 명세하고 이로부터 시험에 필요한 테스트 패턴들을 조직적으로 추출하는 방법을 기술하였다. 본 논문의 테스트 패턴 추출 방법은 프레임워크가 확장될 때 형성 가능한 가변부위의 구조를 객체들간의 연결 형태 기준으로 분류하고, 분류된 각 템플릿에 대한 객체 클러스터를 생성하여 시험하는데 효과적으로 사용될 수 있다. 이를 위해서 추출된 테스트 패턴으로부터 객체 구조 인스턴스를 효율적으로 선정, 또는 자동 생성하는 방법에 대한 연구가 진행중에 있다.

5. 참고 문헌

[1] 전태웅, "객체지향 프레임웍의 Hot Spot에 대한 시험성 강화 방법", 소프트웨어공학 회지, Vol.11, No.4, 1998.

[2] W. Pree, Design Patterns for Object-Oriented Software Development, Addison-Wesley, 1995.

[3] Robert V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison-Wesley, 2000.

[4] P. A. Stocks and D. A. Carrington, "Test templates: a specification-based testing framework", Proceedings of the 15th international conference on Software Engineering, May, 1993, pp. 405-414.

[5] J. Rumbaugh, I. Jacobson and G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 1998.

[6] S. Kirani and W. T. Tasi, "Method Sequence Spedification and Verification of Classes", Journal of Object-Oriented Programming, October 1994, pp. 28-38.

[7] D. Hoffman and P. Strooper, "ClassBench: A Framework for Automated Class Testing", Software-Practice and Experience, May 1997, pp.573-597.

[8] G. Smith, The Object-Z Specification Language, Kluwer Academic, 1999