객체 지향 프레임웍의 가변부위에 대한 상호작용 패턴의 테스트 방법

(Testing of Interaction Patterns for Hot Spots in an Object-oriented Framework)

노 성 환 † 전 태 웅 ††
(Sunghwan Roh) (Taewoong Jeon)

요 약 프레임웍의 기능성을 철저하게 테스팅하기 위해서는 객체 지향 프레임웍의 재사용 시에 확장되는 가변 부위(hot spots)에 대한 체계적인 테스트 패턴 추출이 필수적이다. 본 논문에서는 프레임웍 가변부위의 설계 패턴을 분석하여 가변부위의 상호작용(interaction) 패턴으로부터 테스트 패턴을 추출하는 방법을 제안한다. 프레임웍 가변 부위의 설계 패턴에서 나타날 수 있는 객체들의 상호 작용은 상태도(statechart)로 표현되며, 표현된 상태도는 테스트 패턴 및 테스트 케이스를 생성하는데 사용된다. 생성된 테스트 패턴은 프레임웍을 확장하여 만들어진 어플리케이션들에 반복 적용되어 사용될 수 있다.

키워드 : 객체 지향 소프트웨어 프레임웍, 테스트 패턴, 테스트 케이스, 상태도, 설계 패턴

Abstract Systematically extracting the test patterns of hot spots in an object-oriented software framework is a prerequisite for thoroughly testing the framework's functionality in a variety of contexts in which the framework is extended for reuse. This paper proposes a method for analyzing the design patterns and extracting the test patterns from the interaction test patterns of hot spots in an object-oriented framework. Based on the design pattern of the framework's hot spot, our method captures the object behavior allowed in that hot spot by means of statecharts, which are then used to generate the interaction test patterns and test cases. The generated test patterns and test cases can be applied repeatedly to applications which are built from extending the framework.

Key words : object-oriented software framework, test patterns, test cases, statecharts, design patterns

1. 서 론

현재 유행하는 객체 지향 기술 중 하나는 소프트웨어를 프레임웍으로 개발하는 것이다[1]. 프레임웍은 어플리케이션 계열에서 공통적으로 재사용할 수 있도록 미리 구현된 아키텍처이며, 이러한 프레임웍을 통해 효율적으로 소프트웨어를 개발할 수 있다.

프레임웍은 완전한 어플리케이션 소프트웨어를 생성하기 위해서 인스턴스화되고 수정되는 제네릭(generic)한 소프트웨어 시스템을 말한다. 프레임웍은 그대로 사용할 수 있거나(ready-to-use) 거의 완성된(semi-finished) 빌딩 블록들로 이루어져 있으며 빌딩 블록들 간의 합성 및 상호작용에 대한 규칙과 제약 조건들도 미리 정의되어 있다. 객체 지향 프레임웍은 추상 클래스와 구체 클래스의 집합이 빌딩 블록으로 제공되는 프레임웍이다.

프레임웍은 특정 도메인의 여러 유사한 어플리케이션들로부터 얻어지는 공통적인 면을 나타내게 된다. 따라서 프레임워크를 사용하여 어플리케이션을 개발하게 되면 어플리케이션에 해당되는 특정 부분만큼만 노력이 들게 되며, 코드 콤포넌트 수준의 재사용이 아닌 설계와 코드를 함께 재사용함으로써 대규모의 재사용이 가능하게 된다[2,3].

작은 빌딩 블록들로 구성된 라이브러리(library)를 이용하여 소프트웨어를 재사용하는 전통적인 소프트웨어 재사용 방법에 비해 객체지향 프레임워크는 여러 유사한 어플리케이션의 공통 부분의 추상화를 가장 높은 수준으로 제공한다. 프레임워크를 이용하여 완전한 응용 소프트웨어를 생성할 때는 파라미터화된 프레임워크 컴포넌트

† 비 회 원 : 고려대학교 전산학과
shroh@selab.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터정보학과 교수
jeon@selab.korea.ac.kr
논문접수 : 2004년 7월 6일
심사완료 : 2005년 5월 20일

들을 객체 합성과 클래스 상속 메커니즘을 사용하여 개조, 확장 함으로써 만들어진다. 생성된 소프트웨어 제품은 객체지향 프레임워크의 한 인스턴스라 할 수 있다.

프레임웍은 가변 부위와 고정 부위(frozen spot)가 분리되도록 설계된다. 재사용 시에 개조되는 가변 부위는 프레임웍의 클래스나 메소드의 내부에 캡슐화된다. 고정 부위에 해당하는 클래스(메소드)를 템플릿(template) 클래스(메소드)라고 부르고 가변 부위에 해당하는 클래스(메소드)를 후크(hook) 클래스(메소드)라고 부른다. Pree[4]는 프레임웍의 고정 부위(frozen spot)와 가변 부위(hot spot)에 해당하는 템플릿 클래스와 후크 클래스 사이의 연결 구조 유형을 7가지의 합성 패턴들로 분류하였다.

잘 설계된 프레임웍은 여러 가지 설계 패턴들로 구성된다[5]. 프레임웍을 형성하는 설계 패턴들은 프레임웍의 구조뿐만 아니라 행위를 나타낸다. 또한 대부분의 설계 패턴들은 가변 부위와 고정 부위로 구성되었으며 그들 사이의 상호작용을 잘 나타낸다.

객체지향 프레임웍은 다수의 응용 소프트웨어의 개발에 반복적으로 재사용될 목적으로 만들어지므로 높은 신뢰성이 필수적인 요구 사항이다. 높은 신뢰성을 위해서 프레임웍은 철저한 시험을 거쳐야 한다. 특히, 재사용 시 변경 없이 그대로 사용하는 고정부위보다 재사용할 때마다 컴포넌트들을 개조하고 확장해야 하는 가변 부위에 대한 철저한 시험이 이루어져야 한다.

본 논문에서는 프레임웍 가변부위의 설계 패턴으로부터 상호 작용 패턴을 추출하고 이로부터 테스트 패턴을 체계적으로 생성하는 방법을 제안하였다. 본 논문의 핵심은 설계 패턴의 상호작용 패턴을 상태도로 표현하고 이렇게 표현된 상태도로부터 체계적인 테스트 패턴과 테스트 케이스를 생성하는 것에 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구들을 소개한다. 3장에서는 상태도를 이용하여 프레임웍 가변 부위의 상호작용 패턴을 추출하고 이로부터 테스트 패턴을 생성하는 방법을 설명한다. 4장에서는 샘플 프레임웍에 본 논문의 방법을 적용한 예를 설명한다. 5장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

객체 지향 소프트웨어를 테스트하기 위한 많은 테스팅 방법들이 제안되어왔다. 예를 들어 ASTOOT[6]은 대수 기반의 클래스 테스트 방법과 지원 도구를 제안하였으며, Classbech[7]는 상태 기반의 클래스 테스트 방법과 지원 환경을 제공하였다. 클래스 계층에 대한 점진적인 테스팅[8]과 객체 지향 통합 테스팅 방법[9]도 제안되었다.

Binder[10]는 객체 지향 소프트웨어에 대한 테스트 모델과 테스트 설계 패턴 및 테스트 지원 환경을 포괄적으로 제공하였다. XUnit[11]에서는 클래스의 반복적인 단위 테스팅을 지원하기 위하여 다양한 언어의 테스팅 프레임웍을 제안하였다. 예를 들어 XUnit의 자바 버전인 JUnit[12] 프레임웍은 자바 클래스에 대한 테스트 케이스들을 자동으로 실행할 수 있도록 확장된다. 테스트 데이타 분류 방법(TTF : Test Template Framework)[13]은 소프트웨어의 기능 단위(functional unit)인 오퍼레이션을 테스트하기 위해 고안되었다. TTF는 분할 기준에 의하여 나누어진 입력 데이타 영역에서 테스트 케이스를 선정하도록 설계되었다. [14]에서는 abstract state machine(ASM)으로 작성된 소프트웨어 명세로부터 finite state machine(FSM)를 추출하여 테스트 케이스를 생성하는 방법을 제안하였다. ASM 명세의 행위(action)에 나타나는 Boolean 진입 조건들의 참과 거짓의 경우의 수 중에서 도달 가능한 경우만을 분석하여 FSM를 추출하게 된다.

확장된 프레임웍을 테스팅하는 어려움은 잘 알려져 있으며 몇 가지 해결 방법이 제안되었다[10,15]. Fayad [15]은 프레임웍 테스팅을 위한 BIT(Built-in tests)클래스를 프레임웍 내부에 내장시켜 프레임웍의 상속 및 확장 시에 재사용되도록 하였다. [16]에서는 디자인 패턴들을 포함하는 객체 지향 프레임웍으로부터 개발된 어플리케이션 소프트웨어를 테스트하는 방법을 연구하였으며, 프레임웍으로 개발된 어플리케이션에서 일어날 수 있는 여러 시나리오를 생성하는 시나리오 템플릿을 제안하였다. 시나리오 템플릿으로부터 기존의 테스트 케이스 생성 방법들(파티션 테스트, 랜덤 테스트)을 이용하여 테스트 케이스를 생성하였다. [17]에서는 프레임웍의 가변 부위의 시험성(testability)를 증가시키기 위한 방법을 제안하였다. 테스트 지원 코드를 BIT(built-in test) 컴포넌트로 캡슐화하여 객체 지향 프레임웍의 후크 클래스에 내장함으로써 프레임웍 확장 시에 발생될 수 있는 오류를 효과적으로 알아낼 수 있도록 하였다.

위에서 언급된 테스팅 방법들은 프레임웍 테스팅에 사용될 수 있으나 어플리케이션 개발 시에 확장되는 프레임웍의 가변 부위에 대한 테스트 패턴 생성 문제를 명백하게 설명하지는 않는다. 본 논문에서 제안하는 방법은 객체 지향 프레임웍에서 다른 부분들보다 더욱 많은 변경이 발생하는 가변 부위에 대한 상호작용 테스트 패턴 추출에 직접적으로 초점을 맞추었다는 점에서 위의 논문들과 구별된다.

3. 프레임웍의 가변 부위에 대한 상호작용 패턴 표현과 테스트 패턴 추출

객체 지향 프레임웍 가변 부위의 설계 패턴으로부터 체계적인 테스트 패턴을 추출하기 위하여 다음과 같은 과정으로 상호작용 패턴을 상태도로 표현하고 이로부터 테스트 패턴을 추출한다.

1) 설계 패턴을 구성하는 클래스들의 상태도를 각각 표현한다.
2) 설계 패턴의 상호작용 특성을 나타내는 상태도를 표현한다.
3) 위 1)과 2)에서 나타나는 상태도의 이벤트는 1)의 클래스의 메소드에 해당한다.
4) 가변 부위의 설계 패턴에 대한 상호 작용 패턴을 나타내는 상태도는 1)의 상태도들과 2)의 상태도를 동시 하위 상태(concurrent substate)로 갖는 복합 상태(composite state)로 표현된다.
5) 상호작용을 나타내는 복합 상태를 평면화(flattening)하여 분해한다[10].
6) 평면화된 상태도로부터 전이 트리(transition tree)를 만들어 테스트 패턴을 추출한다[10].

객체지향 프레임웍 가변 부위에 대한 설계 패턴의 하나인 Observer 설계 패턴[5]으로부터 상호작용 패턴을 나타내는 상태도를 추출하는 예를 설명한다. 그림 1의 Observer 설계 패턴에서 Subject와 Observer 클래스는 상대적으로 각각 프레임웍의 고정 부위인 템플릿 클래스와 가변 부위인 후크 클래스에 해당한다. Subject 클래스의 Notify 멤버 함수는 템플릿 메소드이고 Observer 클래스의 Update 멤버 함수는 후크 메소드에 해당한다. Observer 설계 패턴을 갖는 프레임웍을 이용하여 만들어진 어플리케이션은 가변 부위의 Update 함수를 자신의 용도에 맞게 재정의하여 사용한다.

Observer 설계 패턴에서 Subject 객체는 자신의 상태가 변하면 자신과 의존 관계에 있는 모든 Observer 객체들에게 상태 변화를 자동적으로 알린다. Subject 객체의 상태 변화를 알게 된 Observer 객체들은 Subject 객체의 상태에 맞추어 자신의 상태를 갱신하게 된다[5]. 이러한 Observer 설계 패턴의 상호작용 특성은 Subject와 Observer간의 일치성(consistency)이다.

그림 2의 Observer 패턴 객체 구조는 하나의 Subject 객체와 하나의 Observer 객체로 구성되며 그림 3은 그림 2의 객체들 간의 상호작용을 보여준다.

그림 3의 Observer 설계 패턴의 상호작용은 그림 4, 5, 그리고 6의 상태도로 표현된다. 그림 4와 5의 상태도는 Subject 클래스와 Observer 클래스의 행위를 각각 표현한다. 그림 6의 상태도는 Subject 클래스와 Observer 클래스 사이의 일치성을 표현한다.

그림 4, 5는 Z 언어[18]의 형식을 빌려 표현하였다. 그림 4, 5의 주석 부분의 [StateSpace]는 모든 가능한 상태의 집합을 나타내는 given set이다. 그림 4에서 나타나는 subject_states 변수는 Concrete Subject 클래스를 구성하는 상태들을 나타낸다. 그림 4의 current_subject_state와 current_subject_state′ 변수는 전이의 전과 후의 현재 상태를, s? 변수는 외부에서 입력되는 상태를 의미한다. SetState 이벤트를 수신했을 때 발생하는 재귀 전이(self-transition)는 외부에서 입력되는 상태와 현재 상태가 다를 때에만 발생한다. 재귀 전이의 행위(action) 과정 중에서 현재 상태가 입력된 상태로 변경된다. 그림 5에서 나타나는 observer_states 변수는 Concrete Observer 클래스를 구성하는 상태들을 나타낸다. 그림 5의 current_observer_state와 current_observer_state′ 변수는 전이의 전과 후의 현재 상태를, s? 변수는 Concrete Subject 클래스의 상태를 의미한다. Update 이벤트를 수신했을 때 재귀 전이가 발생한다. 재귀 전이의 행위(action) 과정 중에서 현재 상태가 Concrete Subject 클래스의 상태에 맞추어 변경된다. 그림 6의 상태도는 Observer 설계 패턴의 상호작용 성격을 보여준다. Notify와 GetState 이벤트는 불일치(Inconsistency) 상태와 일치(Consistency) 상태 사이의

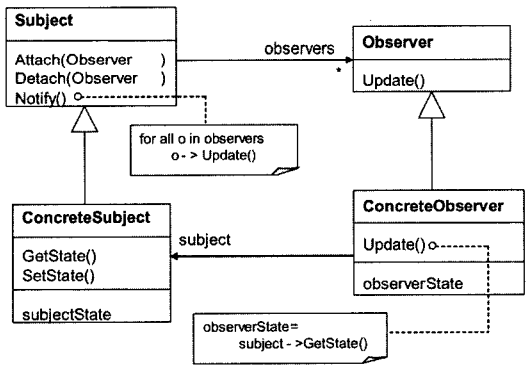

그림 1 Observer 설계 패턴의 구조

그림 2 1:1 Observer 설계 패턴의 객체 구조

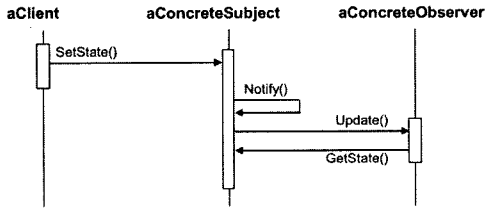

그림 3 1:1 Observer 설계 패턴의 상호작용

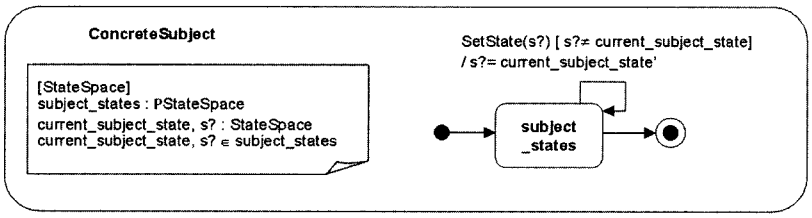

그림 4 Subject 클래스의 행위

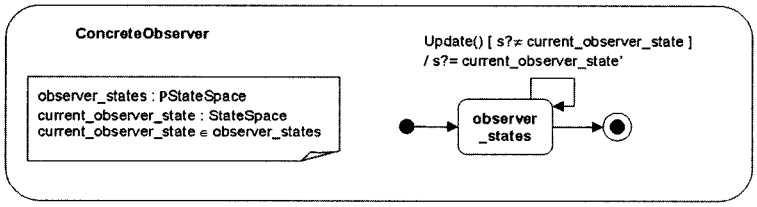

그림 5 Observer 클래스의 행위

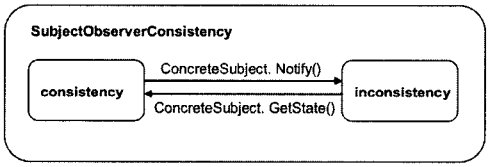

그림 6 Observer 설계 패턴의 상호작용 특성

전이를 발생시킨다.

그림 7의 복합 상태는 그림 2의 1:1 객체 구조의 상호작용 패턴을 나타내며, 그림 4, 5, 그리고 6의 상태를 동시 하위 상태로 갖는다.

Observer 설계 패턴은 Observer 객체의 수에 따라 다양한 형태의 객체 구조를 형성할 수 있다. 여러 개의 Observer 객체로 구성되는 객체 구조의 상호작용을 상태도로 표현할 경우에는 해당 Observer 객체 개수만큼의 상태를 동시 하위 상태로 갖는 복합 상태도로 나타낼 수 있다. 이는 Observer 설계 패턴을 구성하는 각 클래스의 상태와 Observer 설계 패턴의 특성을 나타내는 상태가 분리되었으므로 가능하다.

그림 8의 Observer 패턴을 갖는 1:2 객체 구조는 하나의 Subject 객체와 두 개의 Observer 객체들로 구성된다. 그림 9은 복합 상태는 그림 8의 객체 구조의 상호

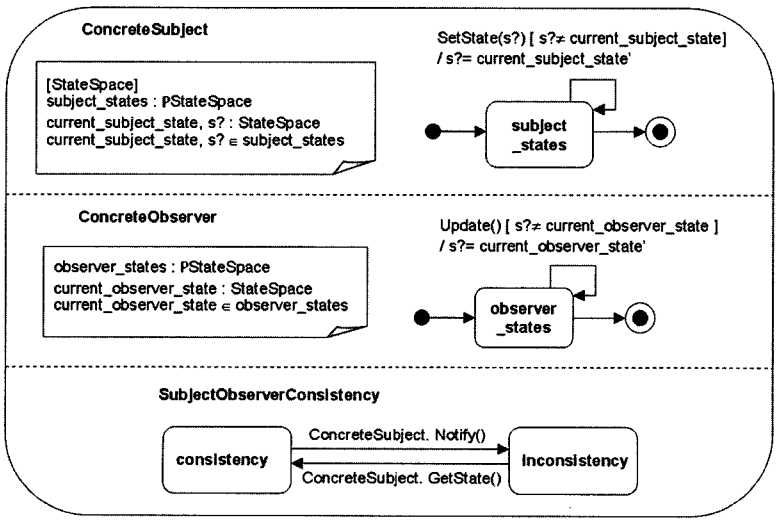

그림 7 1:1 Observer 설계 패턴의 상호작용 패턴

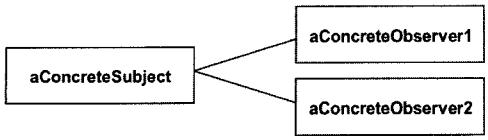

그림 8 1:2 Observer 설계 패턴의 객체 구조

작용 패턴을 보여준다.

본 논문에서는 상태도로 표현된 행위 모델로부터 프레임웍의 가변부위에 대한 상호작용 테스트 패턴을 생성하기 위하여 블랙 박스 테스트 방법을 이용하였다. 객체 지향 프레임웍의 가변 부위에 대한 상호작용 패턴을 표현한 그림 9의 복합 상태도는 테스트 패턴을 생성하는데 사용된다. 복합 상태를 평면화하고 이로부터 전이 트리를 추출하여 테스트 패턴을 생성하게 된다. 테스트 패턴을 추출하는 과정은 다음 사례 연구 절에서 구체적으로 설명된다.

4. 사례 연구

본 절에서는 객체지향 프레임웍 가변 부위에 대한 설계 패턴의 하나인 Observer 설계 패턴을 갖는 경보 감시 프레임웍의 AlaramMonitor 클래스와 AlarmLogger 클래스로부터 상호작용 상태도를 추출하는 과정을 설명한다. 경보 감시 프레임웍의 AlarmMonitor와 AlarmLogger는 각각 Observer 패턴의 Subject와 Observer에 해당된다. 또한 이들은 각각 프레임웍의 고정 부위인 템플릿 클래스와 가변 부위인 후크 클래스에 해당한다. 경보 감시 프레임웍을 확장하여 만들어지는 경보 감시 시스템은 AlarmLogger의 Update 함수를 자신의 시스템의 용도에 맞게 확장하여 사용하게 된다. 경보 감시 시스템의 AlaramMonitor 클래스는 계측치에 대한 상한 또는 하한 값의 초과 여부를 감시하고 AlarmLogger 클래스는 AlaramMonitor 클래스로부터 전달 받은 경보 상태를 상황에 맞게 기록한다. 그림 10의 parameterized collaboration 다이어그램은 Observer 설계 패턴에서 AlaramMonitor와 AlarmLogger 클래스의 역할을 보여준다.

그림 11은 경보 감시 시스템의 설계 구조이다. 그림 12은 AlarmMonitor 클래스의 동적 행위를 단순화하여

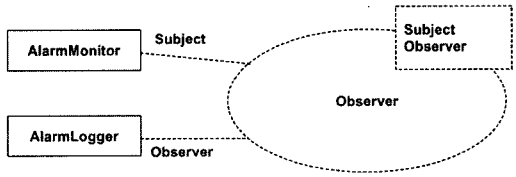

그림 10 Observer 설계 패턴을 갖는 AlaramMonitor와 AlarmLogger

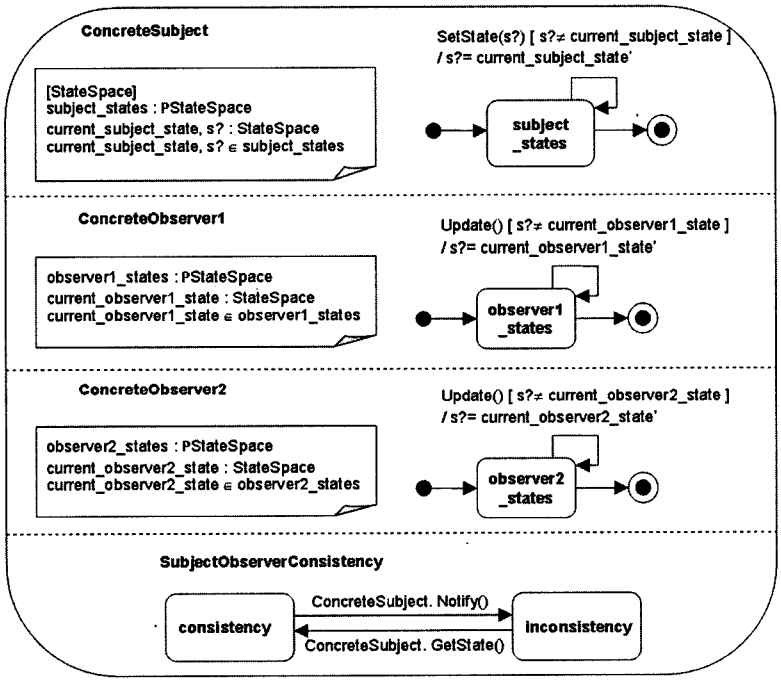

그림 9 1:2 Observer 설계 패턴의 상호작용

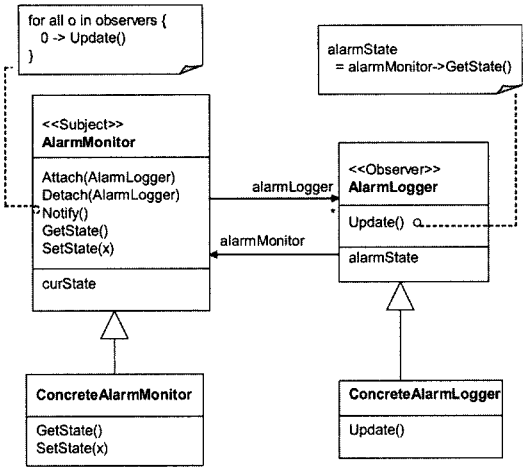

그림 11 경보 감시 시스템의 구조

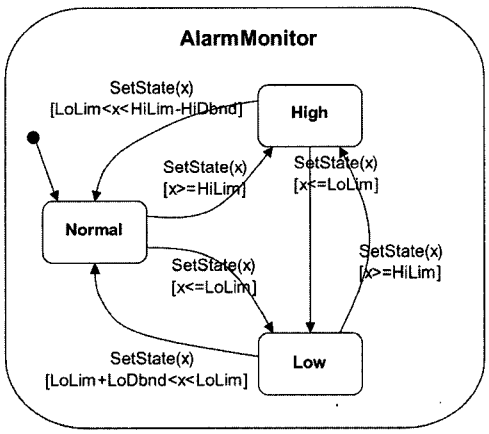

그림 12 AlarmMonitor 클래스의 행위

모델링한 상태도이다. SetState가 호출될 때 패러미터로 입력 받은 계측치가 상한 또는 하한 값을 초과할 경우 AlarmMonitor는 상한 또는 하한 경보를 발생시킨다. AlarmMonitor는 측정치가 상한 또는 하한 경계 값을 오르내릴 때 발생할 수 있는 반복적인 경보를 피하기 위한 범위(deadband)를 가진다.

그림 13의 AlarmLogger 클래스는 AlarmMonitor로부터 Update 호출을 통해 변경된 경보 상태를 넘겨받아 이를 기록한다. 그림 14는 Observer 패턴의 상호작용을 갖는 AlarmLogger 클래스와 AlarmMonitor 클래스 간의 일치성을 나타낸다.

예를 위하여 AlarmLogger와 AlarmMonitor 클래스 객체가 1:1로 연결되는 경보 감시 시스템을 그림 15과 같이 구성한다. 그림 15의 Observer 설계 패턴 객체 구조에 대한 상호작용 패턴은 그림 12, 13, 14를 동시 하

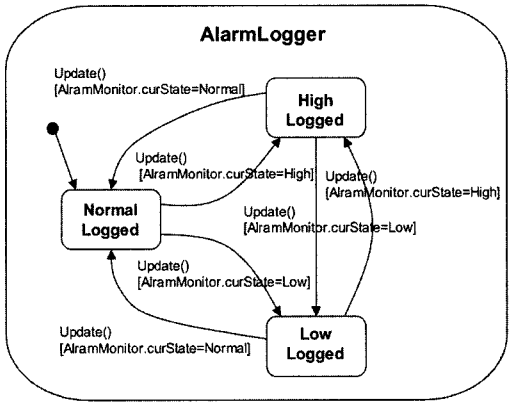

그림 13 AlarmLogger 클래스의 행위

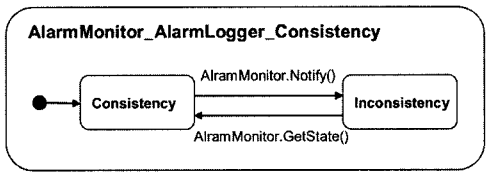

그림 14 AlarmMonitor와 AlarmLogger 간의 일치성

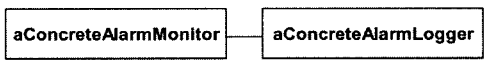

그림 15 경보 감시 시스템의 1:1 객체 구조

위 상태로 갖는 그림 16의 복합 상태로 나타낼 수 있다.

그림 17은 그림 16의 복합 상태를 평면화한(flattened) 상태도다. 그림 17의 평명화된 상태도는 AlamMonitor, AlarmLogger, Consistency의 각 하위 상태의 조합에 의하여 3*3*2=18개의 state를 갖는다. 그림 17의 상태도에서 나타나는 번호는 그림 16의 이벤트에 관련된 번호에 해당한다.

그림 18은 그림 17의 평면화된 상태도에 왕복 경로 방법(round-trip path)[10]을 적용하여 추출되는 전이 트리이다. 추출된 전이 트리는 그림 17의 상태도에 대한 상호작용 테스트 패턴들의 집합을 나타낸다. 그림 18의 전이 트리의 번호는 그림 17의 번호에 해당한다. 각 테스트 패턴은 전이 트리의 루트 노드(root node)에서 시작하여 리프 노드(leaf node)에서 끝나며 그러한 4개의 경로를 갖는다. 테스트 케이스는 각 전이의 이벤트 패러미터 값과 기대 상태(expected state)를 추가함으로써 만들어진다. 그림 18의 전이 트리에서 32개의 테스트 케이스를 얻을 수 있다.

그림 16의 복합 상태는 상호 작용 패턴 표현과 테스트 패턴의 생성 과정을 설명하기 위하여 경보 감시 시스템을 단순화하여 표현한 것이다. 그림 16의 각 하위

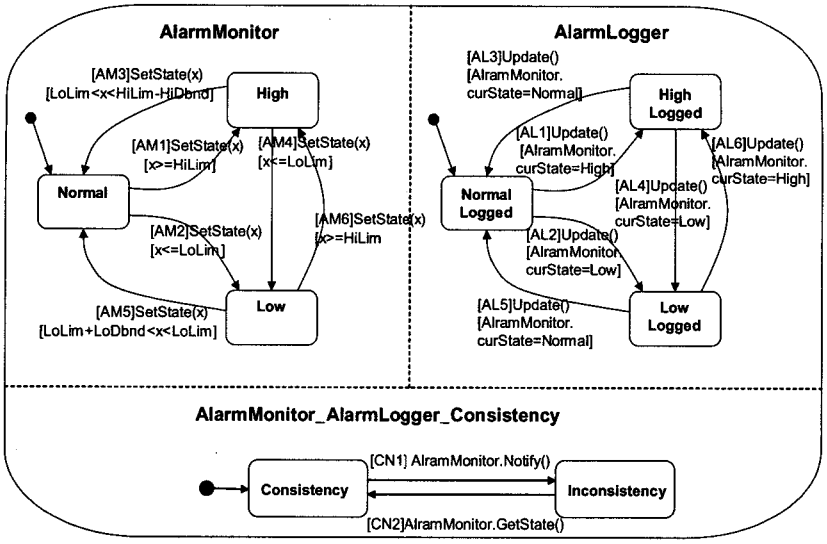

그림 16 1:1 경보 감시 시스템의 상호작용 패턴

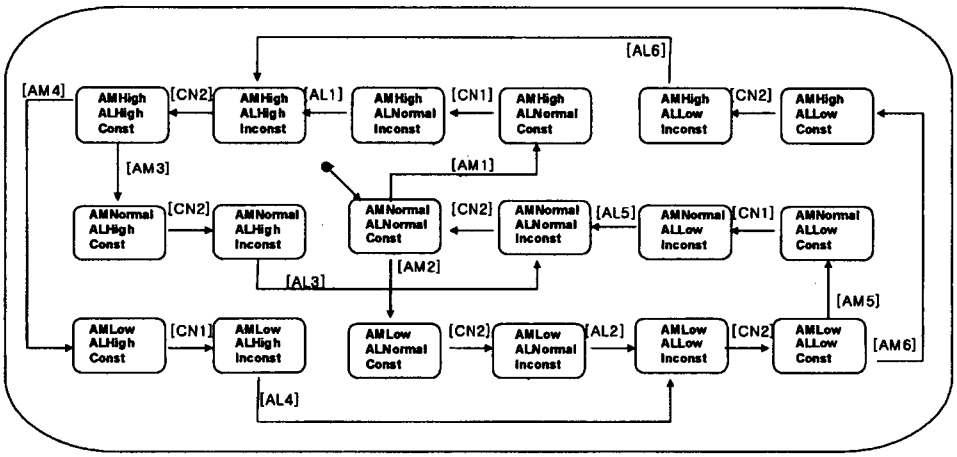

그림 17 그림 16을 평면화한 상태도

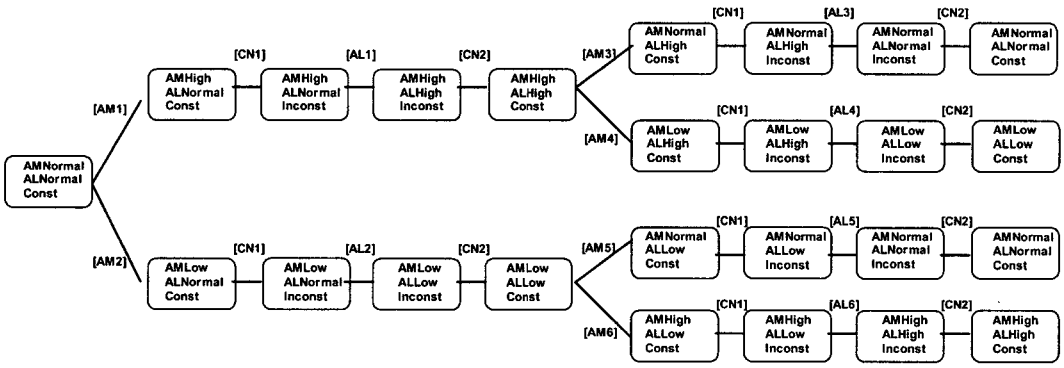

그림 18 그림 17의 평면화된 상태도로부터 만들어지는 전이 트리

상태는 모두 하나의 시작 상태만을 가지며 발생하는 이벤트의 수도 한정되었다. 실제 경보 감시 시스템에서는 시작 상태와 이벤트의 수는 더욱 많을 것이며 이에 따라서 그림 18에서 나타나는 전이 트리의 경로의 수도 크게 증가할 것이다.

생성된 테스트 패턴들은 경보 감시 프레임웍 가변 부위의 AlamMonitor와 AlarmLogger 사이의 상호작용을 표현한다. 이러한 테스트 패턴들은 경보 감시 프레임웍을 확장하여 만들어지는 경보 감시 시스템들을 테스트하기 위한 최소 테스트 케이스 집합으로 사용될 수 있다. 확장된 AlarmLogger의 특정 경보 기록 행위를 위해서는 추가적인 테스트 케이스가 필요하다.

5. 결 론

본 논문에서는 프레임웍의 가변부위에 구현된 설계 패턴을 분석하여 가변부위의 상호작용 테스트 패턴을 추출하는 방법을 제안한다. 프레임웍의 가변 부위의 설계 패턴에서 발생 가능한 객체들의 상호 작용을 상태도로 추출하였으며, 추출된 상태도는 테스트 패턴 및 테스트 케이스를 생성하는데 사용된다. 본 논문은 상호 작용 패턴을 추출하는 예로서 Observer 설계 패턴을 설명하였다. 사례 연구로써 Observer 설계 패턴을 포함하는 경보 감시 프레임웍의 가변 부위에 본 논문의 방법을 적용하여 테스트 패턴과 테스트 케이스를 생성하였으며, 생성된 테스트 패턴들은 프레임웍을 확장하여 만들어지는 경보 감시 어플리케이션에 반복되어 적용될 수 있다.

본 논문에서 제안된 방법은 Observer 패턴 외에 다른 패턴들에도 적용될 수 있다. 향후, 객체 지향 프레임웍의 가변 부위의 다른 설계 패턴들로부터 상호 작용 테스트 패턴을 추출하는 일반적인 방법을 개발하고 이를 지원하는 툴을 개발할 예정이다. 이러한 도구들은 프레임웍의 시험성을 크게 증가시킬 수 있다.

참 고 문 헌

[1] Fayad, M.E., et al., Building Application Frameworks, John Wiley & Sons (1999).

[2] Gregory F. Rogers, "Framework-Based Software Development in C++," Prentice Hall PTR, 1997.

[3] Taligent Inc., "Building Object-Oriented Frameworks," A Taligent White Paper, 1994.

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

[5] Gamma,E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[6] Doong, R. and Frankl, P., "The ASTOOT Approach to Testing Object-Oriented Programs," ACM Trans. Software Eng. and Methodology, Vol. 3, No. 2 (1994) 101-130.

[7] Hoffman, D. and Strooper, P., "ClassBench: a Framework for Automated Class Testing," Software Maintenance: Practice and Experience, Vol. 27, No. 5 (1997) 573-597.

[8] Harrold, M.J., et al., "Incremental Testing of Object-Oriented Class Structures," Proc. 14th Int'l Conf. Software Eng. (1992) 68-80.

[9] Jorgensen, P.C. and Erickson, C., "Object-Oriented Integration Testing," Comm. ACM, Vol. 37, No. 9 (1994) 30-38.

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

[11] The XUnit Home Page, http://www.xprogramming.com/software.htm

[12] Gamma, E. and Beck, K. "JUnit A Cook's Tour," Java Report (1995).

[13] Stocks,P. A., and Carrington,D. A., "Test templates: a specification-based testing framework," Proceedings of the 15th international conference on Software Engineering, May 17-21, 1993, Baltimore, MD, USA, pp. 405-414.

[14] Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte, Margus Veanes, "Generating finite state machines from abstract state machines," ACM SIGSOFT Software Engineering Notes, Volume 27 Issue 4, July 2002.

[15] Fayad, M.E., et al., "Built-In Test Reuse," In the Building Application Frameworks, Fayad, M.E., et al, John Wiley & Sons (1999) 488-491.

[16] Weik-Tek Tsai, Yongzhong Tu, Weiguang Shao, Ebner, E., "Testing extensible design patterns in object-oriented frameworks through scenario templates," Computer Software and Applications Conference, in Proceedings of COMPSAC '99, The Twenty-Third Annual International, 27-29 Oct. 1999 Page(s): 166-171.

[17] Taewoong Jeon, Hyon Woo Seung, Sungyoung Lee, "Embedding built-in tests in hot spots of an object-oriented framework," ACM SIGPLAN Notices, Volume 37 Issue 8, August 2002.

[18] J. B. Wordsworth, Software Development with Z, Addison-Wesley, 1992.

노 성 환

1992년 3월~1997년 8월 고려대학교 컴퓨터정보학과 학사. 1997년 9월~1999년 8월 고려대학교 전산학과 석사. 2000년 3월~2005년 2월 고려대학교 전산학과 박사. 전공분야는 software architecture, software testing, software framework, formal methods 등

전 태 웅

1977년 3월~1981년 2월 서울대학교 계산통계학과 학사. 1981년 3월~1983년 2월 서울대학교 계산통계학과 석사. 1987년 8월~1992년 5월 Illinois Institute of Technology, Dept. Computer Science, Ph. D.. 1983년 3월~1987년 7월 금성통신 연구소 주임연구원. 1992년 9월~1995년 2월 LG산전 연구소 책임연구원. 1995년 3월~현재 고려대학교 컴퓨터정보학과 교수. 전공분야는 소프트웨어 아키텍처, 소프트웨어 테스팅, 객체지향 방법론, 컴포넌트 기반 개발