Design/CPN을 이용한 객체지향 소프트웨어 상호작용 테스트 기법

이 인 혁† · 구 연 설††

요 약

객체지향 시스템은 상호 작용하는 객체들의 집합으로 구성되었고 시스템의 행위는 객체들의 협력 행위로 표현된다. 객체지향 소프트웨어의 상속성, 다형성 등 특성들은 소프트웨어 테스팅에 어려움을 증가시키고, 객체의 병행성, 동적바인딩, 상호작용 등 동적인 면들은 프로그램의 실행 시간에 한 멤버 함수의 호출에 의해 여러 멤버함수로 바인딩될 수 있다. 그러므로 객체지향 소프트웨어의 특성들을 고려하고 동시에 객체들간의 상호작용을 테스트하기 위한 연구가 필요하다. 이 논문에서는 상속성, 다형성 등 특성을 고려하여 평탄화된(Flattened) 상태차트 다이어그램을 구성하고, 시스템 모델링과 시뮬레이션에 전형적으로 사용되는 CPN(Colored Petri net)으로 모델링 한 후, Design/CPN 툴에 적용하여 객체지향 소프트웨어 객체들간의 상호작용 테스트를 자동화할 수 있는 테스트 케이스 생성 기법을 제안한다.

Object-Oriented Software Interaction Test Techniques using Design/CPN

Ren-Ge Li† · Yeon-Seol Koo††

ABSTRACT

An object-oriented system is organized by a set of interacting objects and the system behavior is represented by the cooperating interaction between objects. The characteristics of object-oriented software, such as inheritance and polymorphism, increase the difficulty of the object-oriented software testing. At running time of a program, one call from a member function can bind to other member functions because of the dynamic characteristics such as concurrence, dynamic binding and interaction. Therefore, there need the research about considering the characteristics of object-oriented software and concurrently testing the interaction between objects. In this paper, we propose the techniques as follows. First, we construct a flattened state chart diagram by considering the inheritance and polymorphism. Next, we model the system with CPN(Colored Petri Net) that usually is applying the system modeling and simulation. Last, we propose a test case generation techniques for testing the interaction between objects in object-oriented software by applying a Design/CPN tool.

키워드 : 평탄화된 상태차트 다이어그램(Flattened State Chart Diagram), 칼라 페트리 넷(CPN), 서브투어(Subtour), 디스크립트(Descriptor)

1. 서 론

객체지향 방법론은 재사용성(Reusability), 이식성(Portability), 확장성(Extensibility) 등을 기반으로 많은 분야에서 활용되고 있다. 객체지향 소프트웨어 개발의 주요 관심사는 기존의 클래스에 정의된 데이터나 연산들과 같은 속성들을 상속관계를 통하여 하위 클래스에서 이용하도록 하는 것이다. 만약 기존의 클래스들이 효과적으로 검증되지 않았다면 이미 개발된 클래스들을 다양한 응용분야에 재사용하는 것이 어려울 것이다. 이러한 측면에서 클래스를 설계하거나 개발할 때 클래스의 신뢰성을 확립하는 작업은 필수적이다.

객체지향 소프트웨어를 테스트할 때 상속성, 동적바인딩, 다형성, 객체상호작용 등과 같은 특성들을 고려한 테스팅 기법이 필요하다. 클래스 다이어그램은 객체지향 시스템의 정적인 구조 표현에 사용되는데 클래스들간의 연관성, 일반성, 의존성 등 관계성들을 표현한다. 객체지향 시스템의 동적 행위의 주요한 표현형식은 상태 다이어그램과 상호작용 다이어그램들이다. 상태 다이어그램 형식은 단일 객체들의 라이프사이클에 연관되고 UML의 상태차트 다이어그램과 같은 모델에 의해 일반적으로 명세하는 것이다. 상호작용 형식은 멀티 객체들 간의 상호작용에 관계되고 UML에서 상호작용 다이어그램으로 명세한다.

객체지향 소프트웨어의 방법론은 소프트웨어 개발에 잇점을 가져다주지만 상속성, 다형성 등 특성들은 테스팅에 어려움을 증가시킨다. 객체의 병행성, 동적바인딩, 상호작용 등 동적인 면들은 프로그램의 실행 시간에 한 멤버 함수의

† 준 회 원 : 충북대학교 대학원 전자계산학과
†† 정 회 원 : 충북대학교 전기전자컴퓨터공학부 교수
논문접수 : 2003년 10월 13일, 심사완료 : 2004년 1월 19일

호출이 여러 멤버 함수에 바인딩 될 수 있다는 것을 의미한다. 그러므로 상호 작용하는 객체들의 동적 행위를 테스트하기 위한 연구가 필요하다.

이 논문에서는 객체지향 소프트웨어의 특성들을 고려하고 객체들간의 상호작용에 따른 동적인 테스팅 문제를 해결하기 위하여 객체지향 명세를 특수화된 시각적인 페트리 넷(Petri Net)의 확장버전으로서 시스템 모델링과 시뮬레이션에 전형적으로 사용되는 CPN(Colored Petri net)으로 모델링하고 Design/CPN 툴을 이용하여 객체지향 소프트웨어 객체들간의 상호작용을 테스트하기 위한 테스트 케이스 생성 기법을 제안한다.

객체지향 소프트웨어의 상속성, 다형성 등 특성들에 의하여 UML에서 객체 라이프사이클을 나타내는 상태차트 다이어그램들을 평탄화하여 평탄화된 상태차트 다이어그램으로 확장하고 확장된 상태차트 다이어그램을 정의한 CPN으로의 전환 규칙에 의하여 CPN으로 전환하며 객체들간의 상호작용 명세에 의하여 CPN들을 결합하여 단일 CPN으로 구성한다. 구성된 CPN을 Design/CPN 툴에 적용하여 시뮬레이션과 분석을 진행한다. 그 결과 생성된 도달성 그래프와 디스크립트 분석을 통하여 서부투어를 생성한다. Design/CPN을 이용한 테스트 케이스 생성을 통하여 제어흐름은 물론 자료 흐름, 병행성 문제까지 고려할 수 있고 데드락과 같은 테스트 케이스의 실행 불가능한 비결정적 문제를 해결할 수 있으며 테스트 케이스 자동화 생성을 위한 기반을 마련한다.

2장에서 객체지향 소프트웨어의 특성들이 소프트웨어 테스팅에 미치는 영향과 CPN에 대해 요약하여 기술하고 기존의 객체지향 소프트웨어 테스팅 기법을 비교 분석한다. 3장에서 상태차트 다이어그램을 평탄화하여 정의한 CPN으로의 전환 규칙에 의한 CPN으로의 전환을 기술한다. 4장에서는 Design/CPN 툴에 적용하여 테스트 케이스를 생성하는 기법을 기술한다. 5장에서 결론 및 향후 연구 방향을 제시한다.

2. 관련 연구

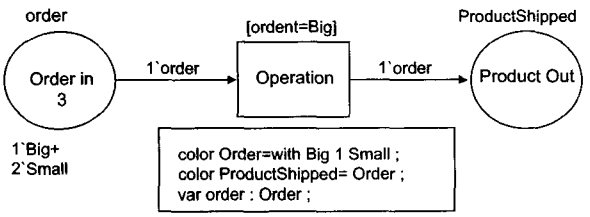
2.1 객체지향 소프트웨어의 특성과 테스팅

객체지향 소프트웨어의 방법론은 소프트웨어 개발에 잇점을 가져다주지만 상속성, 다형성 등 특성들은 테스팅에 어려움을 증가시킨다. 객체지향 소프트웨어 상속성은 이미 테스트가 완료된 클래스로부터 상속된 멤버 함수들 중에서 어떤 멤버 함수들을 하위 클래스 환경에서 다시 테스트 할 것인가를 결정하는 문제를 발생시키고[1], 동적바인딩과 다형성은 프로그램의 실행 시간에 한 멤버 함수의 호출에 여러 멤버 함수로 바인딩 될 수 있다는 것을 의미한다. 이 때문에 주어진 멤버 함수의 호출이 실제적으로 어떤 멤버 함수로 바인딩 된다는 사실을 정적으로는 알 수 없다. 객체들간의 상호작용은 메시지를 통하여 이루어진다. 메시지에 의한 클래스들의 멤버 함수들에 대한 통합 테스팅 경우에 정적인 호출 순서가 존재하지 않으므로 체계적인 기준을 설정한 후, 유한한 개수의 호출 순서에 대하여 테스팅을 하여야 한다. 따라서 정적분석을 통하여 테스팅 정보를 얻는 테스팅 방법들은 이러한 불확실성을 다룰 수 있는 수단을 제공하여야 한다[2].

2.2 CPN(Colored Petri net)

페트리 넷은 시스템을 분석하고 설계하는 모델링 기법으로서, 객체와 정보의 흐름을 다루는 복잡한 분산시스템 설계와 분석을 용이하게 한다[3]. 페트리 넷은 플레이스와 트랜지션을 시스템의 특성에 맞게 아크(arc)로 연결하여 이벤트와 조건을 연결시키면서 플레이스에서의 토큰의 변화를 통해 시스템의 동적 상태를 기술한다. 이벤트의 발생은 특정 트랜지션의 점화(firing)를 통해 표현한다. 페트리 넷에서 시스템의 상태는 마킹(making)으로 표현되는데 마킹은 어떤 플레이스에 얼마만큼의 토큰이 있는가를 나타낸다. 시스템의 상태공간은 도달성 그래프를 통해 나타낸다. 도달성 그래프는 모든 마킹에 대해 어떤 마킹에서 어떤 트랜잭션이 점화해서 어떤 마킹이 되는지를 나타낸다.

그러나 페트리 넷은 제어흐름만을 고려하므로 토큰이 단일 칼라(color)를 가질 뿐이므로 자료흐름까지 고려한 CPN(Colored Petri Net)이 Jansen에 의해 제안되었다[4]. CPN은 페트리 넷의 확장형으로서, 토큰들은 값을 갖고 칼라로 분류되는 것이며 칼라들에 대한 표현식은 전이들과 연관되어 자료흐름까지 고려할 수 있다. 이것은 또한 시스템 설계, 명세, 시뮬레이션, 검증 등을 수행하는 그래픽 중심의 언어로서, 기존의 네트워크 프로토콜의 적합성 테스팅과 시스템 협력 분석에서 CPN을 이용하였다. (그림 1)은 CPN의 예를 나타낸 것이다[5].



(그림 1) CPN의 예

(그림 1)에서 보는 바와 같이 CPN은 플레이스, 전이와 아크를 포함하고 각각 원형, 사각형(혹은 바), 아크로 표시한다. 마킹은 플레이스에서의 토큰들에 대한 일종의 맵(map)이고 토큰들의 이동은 상태들의 전이를 나타낸다. 전이에 대한 입력 플레이스는 적어도 하나의 토큰을 가져야 하고 전이가 점화하면 토큰들은 출력(output) 플레이스들로 이동하며 이동은 시스템 상태 전이들과 일치하다. CPN에서 우리는 토큰에 속성들을 첨부할 수 있어 칼라 토큰이라고 부른다.

토큰들의 속성은 속성 값의 타입들과 같이 칼라로 정의하는 것이다. 표현식(expression)은 토큰의 흐름을 제어하고 전이에서 가드를 서술하여 전이 점화를 제어할 수 있다.

전이에서 다음의 조건을 만족하면 점화할 수 있다[6].

① 각 입력 플레이스들은 적어도 하나의 토큰을 가져야 한다.
② 전이 입력 아크들에 첨부된 표현식을 입력 토큰들이 만족하여야 한다.
③ 전이에 첨부된 가드 조건을 만족하여야 한다.

Design/CPN은 CPN의 사용을 지원하는 툴 패키지로서, 복잡한 데이터 타입 및 데이터 처리를 하는 CPN 모델을 지원하고 잘 정의된 인터페이스를 가지는 분리된 모듈, 즉 계층적 CPN을 지원한다. Design/CPN은 편집기, 시뮬레이터, OG(Occurrence Graph) 툴로 구성된다. 편집기는 CPN 모델의 구성, 수정, 구문 체크를 지원하고, 시뮬레이터는 CPN의 인터랙티브 및 자동 시뮬레이션을 지원하며, OG 툴은 도달성 그래프와 디스크립트 생성을 지원하는데 이를 분석하여 테스트 케이스 생성에 활용할 수 있다.

2.3 객체지향 소프트웨어 테스팅 기법

기존의 객체지향 소프트웨어의 상속성, 다형성과 동적행위 테스팅을 위한 연구 방법들을 비교분석한다.

Kim의 상태 기반 테스팅은 상속성과 다형성에 따른 테스트 케이스를 생성하기 위하여 분할 상태도인 PS를 만들어 실행하였다[7]. 그러나 이 기법은 테스트 케이스 생성 자동화 실현이 어렵고 또한 테스트 케이스 실행 가능성을 고려하지 않았다.

계층 구조의 점진적 테스팅 기법은 상속성 테스트를 위하여 상위 클래스를 테스트 한후, 하위 클래스를 설계하고 테스트 할 때 변경된 부분만을 선택하여 테스트 하는 방법을 제안했다[8]. 이 기법은 잘 설계된 상속 계층 구조에는 적합하나 객체들간의 상호작용에 대한 테스트에는 유용하지 못하다.

Hong은 클래스 테스팅을 위하여 CSM(Class State Machine)을 정의한 후 CFG(Class Flow Graph)에 적용하여 테스트 케이스를 생성한다[9]. Hong의 기법은 상속성과 다형성 등 객체지향 특성들을 고려하지 않았다.

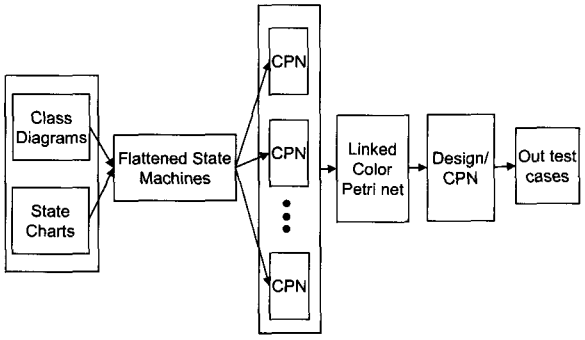
FREE(Flattened Regular Expression) 모델의 상태 기반 테스팅은 수많은 연구결과들의 응용이다[10]. 단위 테스팅으로부터 시스템 테스팅에 이르기까지 FREE 모델의 적용을 시도하였고 상속 관계, 다형성 등 거의 모든 것을 광범하게 다루었으나 객체간의 상호작용에 대한 연구가 미흡하다.

Lee의 Design/CPN을 이용한 프로토콜 테스트 케이스 생성 기법[11]은 단일 모듈에 국한 되었으며 테스트 케이스 실행 가능성에 대하여 다루지 않았다.

그러므로 객체지향 소프트웨어의 상속성, 다형성, 동적바인딩 등 특성을 고려하고 객체들간의 상호작용을 효과적으로 테스트하기 위한 연구가 필요하다.

3. 객체지향 소프트웨어 CPN으로의 전환

객체지향 소프트웨어의 상속성, 다형성 등 특성들을 고려하고 계층구조를 나타내는 클래스 다이어그램에 의하여 객체 라이프사이클을 나타내는 슈퍼 클래스의 상태차트 다이어그램을 평탄화하여 서브 클래스들의 평탄화된 상태차트 다이어그램을 구성한다. 그다음 각 상태차트 다이어그램을 전환 규칙에 의하여 CPN으로 전환하고, 객체간의 상호작용과 연관관계에 의하여 단일 CPN을 구성한다. 구성된 CPN을 Design/CPN 툴에 적용하여 서브투어와 디스크립트 생성을 통하여 테스트 케이스를 생성한다. (그림 2)는 테스트 케이스 생성과정을 그림으로 나타낸 것이다.



(그림 2) 테스트 케이스 생성 흐름도

3.1 평탄화된 상태차트 다이어그램

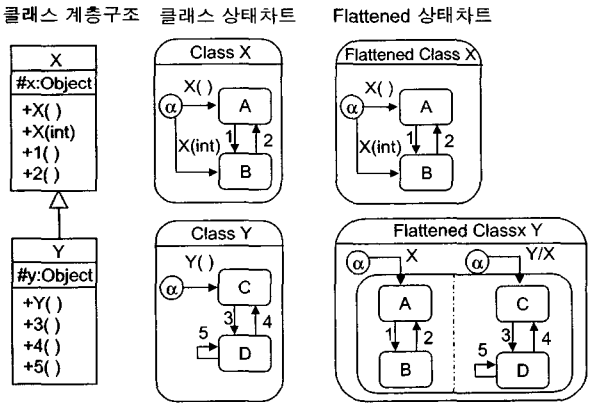
객체지향 소프트웨어에서 클래스들은 계층구조를 가지며 상속성, 다형성, 연관성과 같은 특성들에 의해 연결되는 것이고, 객체 라이프사이클을 나타내는 상태차트 다이어그램들은 객체들의 행위를 나타낸다. 객체지향 소프트웨어의 상속성, 다형성, 동적 바인딩 등 특성들 때문에 보내온 메시지에 대해 어느 메소드가 실행되는지 정적으로 알 수 없다. 상속성과 집단성에 관련된 모두가 다형성이고 동적 바인딩이다. 상속성과 집단화구성 관계성들은 다형성과 결합하여 객체들간의 상호작용 테스팅에서 결함 발견에 어려움을 증가한다.

상속성은 서브 클래스가 슈퍼 클래스의 속성들과 메소드들을 상속받게 되고 서브 클래스에서 새로운 속성과 메소드를 추가하거나 재정의할 수 있다. 객체지향 소프트웨어에서 다형성은 복수의 클래스가 하나의 메시지에 대해 각 클래스가 가지고 있는 고유한 방법으로 응답할 수 있는 능력을 말한다.

UML의 상태차트 다이어그램들은 상속성, 다형성 등 특성에 의해 직관적으로 상태와 전이들을 나타내지 못한다[12, 13]. 그러므로 상태 차트 다이어그램을 CPN으로 전환하기 위하여 우선 객체지향 소프트웨어의 상속성과 다형성

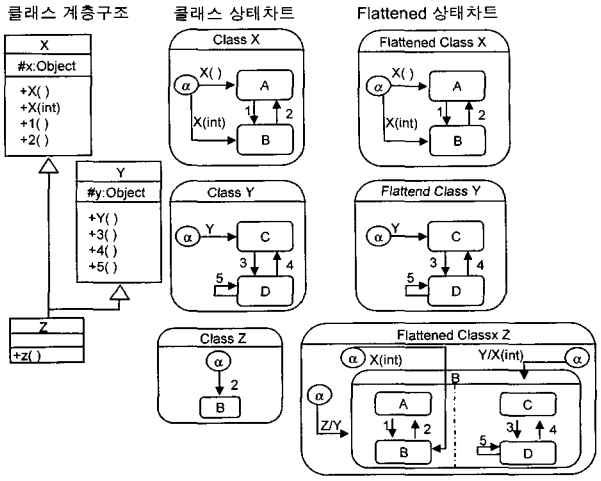
을 고려하여 슈퍼 클래스의 상태차트 다이어그램을 평탄화하여 서브 클래스의 평탄화된 상태차트 다이어그램을 구성한다. 기존의 상태차트 다이어그램 평탄화 연구에서는 상속구조에 근거하여 전이 Splitting, 전이 Retargeting, 상태 분할과 서브 상태 추가, 연결(concatenation), Orthogonal 구성(composition)으로 분류하여 평탄화된 상태차트 다이어그램을 구성하였다[13]. 이 논문에서는 상속계층에서 상속과 재정의 부분을 명확히 하고 또한 다형성의 특성을 고려하여 서브 클래스에 재정의 없는 경우, 재정의 있는 경우, 상속만 존재하는 경우와 다형성의 경우로 분류하여 평탄화된 상태차트를 구성한다.

① 재정의 없는 경우 : 서브클래스가 슈퍼클래스의 특징(feature)들을 상속받고 재정의가 없이 일부 새로운 지역(local) 특징들을 정의하는 경우, 서브클래스의 평탄화된 상태차트 다이어그램은 슈퍼클래스의 상태를 그대로 유지하면서 서브클래스 지역 특징들에 의한 서부 상태들의 추가로 구성한다. (그림 3)은 서브클래스에 재정의가 없는 경우를 그림으로 나타낸 것이다. 그림에서 서브클래스 Y는 슈퍼클래스 X의 특징들을 상속받고 재정의 없이 새로운 특징들을 정의하였다. 이 경우에 슈퍼클래스의 상태 A, B를 유지하면서 서브 클래스의 새로운 특징들의 정의에 의해 상태 C, D를 추가하여 서브클래스의 평탄화된 상태차트 다이어그램을 구성하는 것이다.



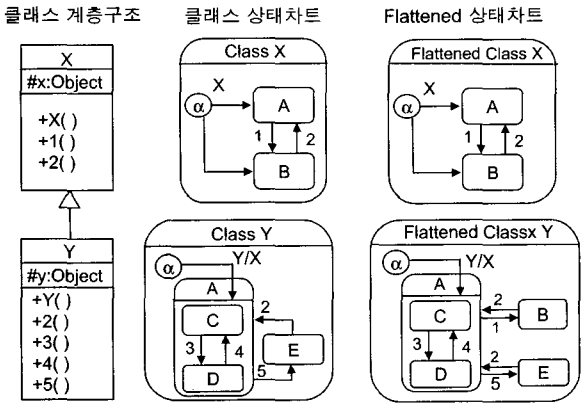
(그림 3) 평탄화된 상태차트 뷰(재정의 없는 경우)

② 상속만 존재하는 경우(단일 상속 혹은 다중 상속) : 서브클래스에 지역 특징들의 정의가 없이 슈퍼클래스들의 특징들만을 상속하는 경우, 서브클래스의 평탄화된 상태차트 다이어그램은 슈퍼클래스들의 상태차트 다이어그램 연결로 구성한다. (그림 4)는 서브클래스에 지역 특징들 정의가 없이 상속만 존재하는 경우를 그림으로 나타낸 것이다. 그림에서 서브클래스 Z는 지역 특징들 정의가 없이 슈퍼클래스 X와 Y의 특징들을 상속받으므로 서브클래스의 평탄화된 상태차트 다이어그램은 슈퍼클래스 X의 상태 A, B와 슈퍼클래스 Y의 상태 C, D의 연결로 구성하는 것이다.



(그림 4) 평탄화된 상태차트 뷰(상속만 존재하는 경우)

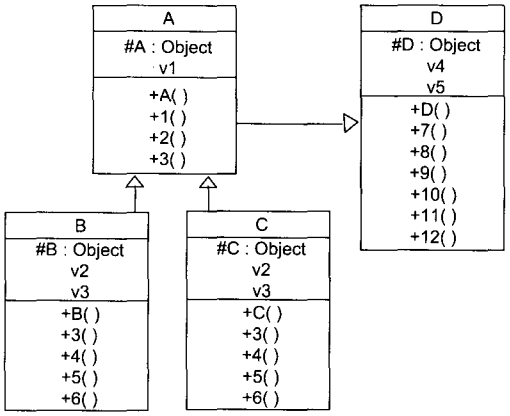
③ 재정의 있는 경우 : 서브클래스에서 새로운 지역 특징들을 정의하고 슈퍼클래스의 일부 메소드들의 오버라이딩이 있는 경우 서브클래스의 평탄화된 상태차트 다이어그램은 슈퍼클래스의 상태 분할과 서브클래스의 지역 특징들 정의에 의한 서부 상태 추가로 구성한다. (그림 5)는 서브클래스에 재정의가 존재하는 경우를 그림으로 나타낸 것이다. 서브클래스 Y는 슈퍼클래스 Y 특징들을 상속받고 메소드 2()를 재정의 하였다. 이 경우에 서브클래스 Y의 평탄화된 상태차트 다이어그램은 클래스 X의 상태 B의 유지, 상태 A의 C, D로의 분할과 새로운 서브 상태 E의 추가로 구성하는 것이다.



(그림 5) 평탄화된 상태차트 뷰(재정의 있는 경우)

④ 다형성의 경우 : 클래스들이 상속 계층구조를 갖고 서브클래스들이 다형성 관계를 갖는 경우에 슈퍼클래스와의 재정의 없는 경우, 재정의 있는 경우, 상속만 하는 경우에 따른 정의 ①, ②, ③에 근거하여 각자 서브

클래스들의 평탄화된 상태차트 다이어그램들을 구성한다. 다형성에 의한 각 서브클래스들의 평탄화된 상태차트 다이어그램들의 연결은 CPN으로 전환한 후, 단일 CPN으로의 구성을 통하여 이루어진다. 다형성에서 서브클래스의 비결정적인 실행은 CPN에서 가드 조건으로 제어한다.



(그림 6) 클래스 다이어그램

(그림 6)은 추상적으로 계층 구조의 클래스 다이어그램을 나타낸 것이다. 시스템은 4개 클래스 A, B, C, D로 구성되고 클래스 B와 C는 클래스 A를 상속받고, 클래스 B와 C는 다형성 관계를 가지며 클래스 A, B, C는 클래스 D와 연관관계를 가진다.

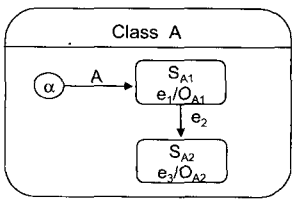
상태차트 다이어그램은 객체의 라이프사이클 동안 연산의 실행과 상태 전이를 명세한다. 각 상태는 노드에 의해 나타내고 상태들의 각 전이는 직선 아크에 의해 나타내며 라벨된 이벤트가 전이를 트리거함을 보여주고, 연산은 실행되는 것이다.

상태차트 다이어그램에 의해 나타내는 라이프사이클 모델은 6개의 튜플(S, E, O, T, X, V) 형식에 의해 표시하는데 S는 상태들의 집합, E는 이벤트들의 집합, O는 연산들의 집합, T는 상태들 전이 집합, X는 연산들의 실행 집합, V는 메시지 흐름(전달인자(parameter))이다.

상태 S_1에서 상태 S_2까지 전이는 3개 튜플(e, s_1, s_2)에 의해 표시하는데 e는 트리거링 이벤트이다.

연산 O의 실행은 2개 튜플(e, o)에 의해 표시하는데 e는 트리거링 이벤트이다.

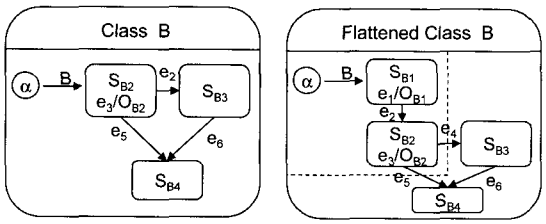
(그림 7)은 클래스 A의 상태차트 다이어그램을 나타낸 것이다.



(그림 7) 클래스 A의 상태차트 다이어그램

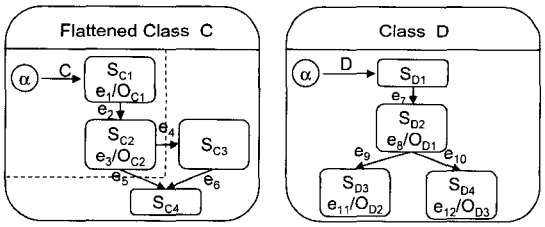
클래스 B와 C는 클래스 A의 메소드 1과 2를 상속받고 메소드 3을 재정의하였으며 새로운 메소드 4, 5, 6을 추가하였다. 클래스 B와 C의 평탄화된 상태차트 다이어그램은 클래스 A의 상속받은 부분들을 참조하고 또한 추가된 속성들과 메소드들을 고려하여 상태의 추가 등의 과정을 거쳐 위에서 정의한 규칙에 의하여 평탄화된 상태차트 다이어그램을 구성한다.

(그림 8)은 클래스 B의 상태차트 다이어그램과 평탄화된 상태차트 다이어그램을 나타낸 것이다.



(그림 8) 클래스 B의 상태차트 다이어그램과 평탄화된 상태차트 다이어그램

같은 방법으로 다형성 관계를 갖는 클래스 C의 평탄화된 상태차트 다이어그램을 구성한다. (그림 9)는 클래스 C의 평탄화된 상태다이어그램과 클래스 D의 상태차트 다이어그램을 나타낸 것이다.



(그림 9) 클래스 C의 평탄화된 상태차트 다이어그램과 클래스 D의 상태차트 다이어그램

3.2 평탄화된 상태차트 다이어그램 CPN으로의 전환

객체지향 소프트웨어 명세를 CPN으로 모델링하여 Design/CPN 툴에 적용하기 위하여 평탄화된 상태차트 다이어그램을 CPN으로 전환하는 규칙을 정의한다.

3.2.1 상태차트 다이어그램과 CPN의 형식적 표기

객체지향 기법에서 시스템은 상호 작용하는 객체들의 집합으로 구성되었고 시스템의 행위는 객체들의 협력 행위로 표현되며 객체들의 상호작용은 메시지 전달에 의해 이루어진다. 상태차트 다이어그램에 의해 나타내는 라이프사이클 모델은 6개의 튜플(S, E, O, T, X, V) 형식으로 구성한다.

$$F = (S, E, O, T, X, V)$$

여기서

S : 유한한 상태(s_1, s_2, …, s_n)의 집합

E : 유한한 이벤트(e_1, e_2, …, e_n)의 집합
O : 유한한 연산의 집합
T : 전이 함수
X : 유한한 출력의 집합
V : 메시지 전달(전달인자 혹은 태스크의 집합 = Param eter∪Task를 포함)

CPN은 페트리 넷에 칼라의 개념을 추가하여 다음과 같이 6개의 튜플(P, T, C, I, O, μ)로 구성된다.

$$CPN = (P, T, C, I, O, \mu)$$

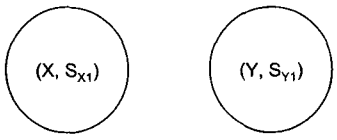
여기서
P : 플레이스의 집합 =(p_1, p_2, … p_n) $n \geq 0$
T : 트랜지션의 집합= (t_1, t_2, … t_n) $n \geq 0$
C : 칼라(color) 함수, C(p)=k'c, k≥0이고 c는 칼라
I : 입력 함수
O : 출력 함수
$\mu(P \rightarrow)(0, 1, \cdots)$: 토큰의 초기 상태 집합

$$P \cap T = 0 \text{ and } P \cup T \neq 0$$

3.2.2 상태 차트 다이어그램 CPN으로의 전환 규칙

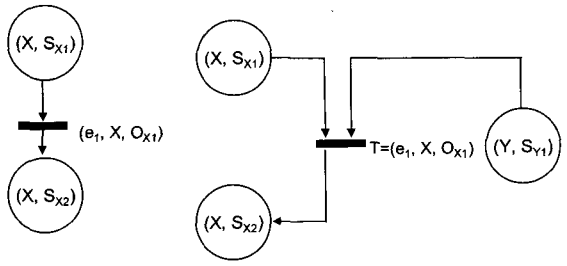
이 부분에서 평탄화된 상태차트 다이어그램을 CPN으로 전환하기 위하여 다음과 같은 규칙을 정의한다.

① CPN 플레이스(P)의 정의 : 상태차트 다이어그램에서 객체의 각 상태는 CPN의 플레이스로 표시한다. CPN 플레이스는 두개 튜플(x, S_x)로 표시하는데 x는 객체이고 S_x는 객체 x의 상태를 나타낸다. 서로 다른 객체들은 서로 구별되는 칼라 플레이스를 갖는다. (그림 10)은 객체 x가 상태 S_{x1}, 객체 y가 상태 S_{y1}, 있음을 나타낸 것이다.



(그림 10) 객체 x와 y의 상태 플레이스

② CPN 전이(T)의 정의 : 상태차트 다이어그램에서 상태들의 각 전이는 연산의 실행과 마찬가지로 CPN의 전이로 표시한다. CPN 전이는 3개 튜플(e, x, o_{x1})에 의해 표시하는데 여기서 e는 트리거 이벤트, x는 객체, o_x는 객체 x의 연산이다. 전이는 자체의 이벤트에 의해 발생하거나 혹은 다른 객체의 메시지를 받아 발생될 수 있다. 메시지는 전달 인자를 포함할 수 있다. (그림 11)은 다른 상태로의 전이를 나타낸 것이고 (그림 12)는 자체 상태로의 전이를 나타낸 것이다.

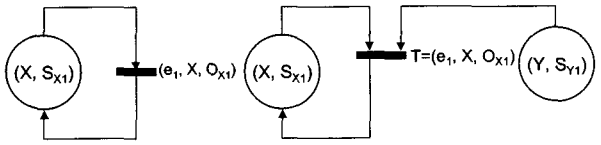


(a) 상태 전이 (b) 상호작용에 의한 상태 전이

(그림 11) 다른 상태로의 전이

(그림 11)(a)는 객체 x의 상태 S_{x1}에서 S_{x2}로의 전이를 CPN 전이(e_1, x, o_{x1})로 보여주는데 연산 o_{x1}은 이벤트 e_1에 의해 트리거되어 실행되는 것이다.

(그림 11)(b)는 객체 x의 상태 S_{x1}에서 S_{x2}로의 전이를 CPN 전이(e_1, x, o_{x1})로 보여주는데 이벤트 e_1은 객체 y로부터의 메시지에 의해 트리거되어 연산 o_{x1}의 실행이 이루어진다.



(a) 상태전이 (b) 상호작용에 의한 상태전이

(그림 12) 자체 상태로의 전이

(그림 12)(a)는 객체 x가 상태 S_{x1}에서 연산 o_{x1}의 실행을 보여주는데 트리거 이벤트는 e_1이다.

(그림 12)(b)는 CPN의 전이(e_1, x, o_{x1})는 상태 S_{x1}에서 객체 x의 연산 o_{x1}의 실행을 보여주는데 이벤트 e_1은 객체 y로부터의 메시지에 의해 트리거 된다.

③ 칼라(C) 토큰의 정의 : 칼라 토큰은 실행을 위해 CPN에 할당 된다. (그림 10)의 플레이스 (x, S_{x1})는 하나의 객체의 토큰을 갖고 객체 x가 상태차트 다이어그램에서 상태 S_{x1}에 있음을 의미한다. 상호작용하는 객체들은 서로 다른 칼라 토큰으로 표현한다. CPN에서 메시지 전달에 의한 입력과 출력 토큰들은 상태와 구별되는 칼라 토큰으로 정의하고 데이터 타입들, 데이터 객체들, 변수들을 사용하여 속성을 구성할 수 있고 데이터 값들을 갖는다. 아크 입력/출력에 표현식을 서술하여 토큰들의 흐름을 제어할 수 있고 전이 조건에 토큰들의 속성 값을 부여하여 전이 점화를 제어할 수 있으므로 상속성, 다형성에 의한 비결정적인 문제를 해결할 수 있다.

④ 입력함수 I는 현재 객체 상태 플레이스 토큰과 상호작용 객체에 의한 메시지 전달 토큰(V)을 입력으로 하여 점화한다. 트랜지션에서 출력 함수인 O는 다음 상태에 토큰을 전송한다.

⑤ 마킹의 정의 : CPN의 마킹 μ는 상호작용 객체들의 현재 상태 서술한다. 처음의 마킹 μ_w은 상호작용 객체

들의 처음 상태와 메시지 전달을 위한 전달 토큰(V)을 서술한다. 마지막 마킹 μ_w는 상호작용 객체들의 마지막 상태를 서술한다.

⑥ 점화 규칙 : CPN의 전이 (e, x, o_x)는 전이 조건을 만족하면 점화한다. 전이가 발생하면 각 출력 플레이스는 CPN 전이로부터 토큰을 받게 된다. 전이 점화는 이벤트 e가 발생하고 객체 x의 연산 o_x가 트리거되어 실행되었음을 의미한다.

⑦ 실행 : CPN의 실행은 상호작용 객체들의 주어진 이벤트들의 연속 발생에 의해 실행된다. 토큰들은 최초 마킹에 일치하게 할당되고, 이벤트들은 주어진 발생의 연속에 일치하여 발생된다. 전이는 처음 이벤트 발생에 의해 점화하고 실행은 마지막 이벤트가 발생한 다음 정지한다. 이때 CPN은 마지막 마킹 μ_w를 갖고 객체들의 마지막 상태를 표시한다.

상태차트 다이어그램을 CPN으로의 전환 알고리즘은 다음과 같다.

① 상태차트 다이어그램의 상태와 입/출력을 플레이스로 사상시킨다.

② 입력 플레이스는 입/출력 심벌과 현재 상태에 해당하는 플레이스이고 출력 플레이스는 다음 상태에 해당하는 플레이스를 정의한다.

③ 초기상태 플레이스에 객체를 나타내는 칼라를 가진 토큰을 위치한다. 셀프루프로 점화 시에는 입/출력 플레이스를 갖지 않고 점화한다.

④ 각 입/출력 플레이스에 토큰을 위치시킨다. 이때, 토큰은 각 전이의 입/출력에 따라 칼라가 부여한다.

⑤ 각 플레이스의 outdegree인 아크에는 토큰 칼라에 따라 라벨을 부여한다.

⑥ 각각의 상태와 입/출력에 대하여 단계 1부터 단계 5의 과정을 반복한다.

3.3 CPN의 구성

상태차트 다이어그램에 의해 나타내는 객체 라이프사이클 모델은 6개의 튜플(S, E, O, T, X, V) 형식에 의해 표시한다. 클래스 B와 D는 연관관계를 갖는다. 우리는 객체 B의 이벤트 e_1과 e_3의 발생은 각각 객체 D의 s_{D1}과 s_{D2}에서의 메시지 전달 v4, v5에 의해 점화하고, 객체 D의 이벤트 e_7, e_9, e_{10}의 발생은 각각 객체 B의 s_{B2}와 s_{B4}에서 메시지 전달 v1, v2, v3에 의해 점화한다고 가정한다. 평탄화된 클래스 B의 객체 라이프사이클 모델을 이 형식으로 표시하면

$S = \{s_{B1}, s_{B2}, s_{B3}, s_{B4}\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

$O = \{o_{B1}, o_{B2}\}$

$T = \{(e_2, s_{B1}, s_{B2}), (e_4, s_{B2}, s_{B3}), (e_5, s_{B2}, s_{B4}), (e_6, s_{B3}, s_{B4})\}$

$X = \{(e_1, o_{B1}), (e_3, o_{B2})\}$이다.

$V = \{v1, v2, v3\}$

v1, v2, v3 : 객체 D의 이벤트 e_7, e_9, e_{10} 발생을 위한 전달인자

클래스 D의 객체 라이프사이클 모델을 이 형식으로 표시하면

$S = \{s_{D1}, s_{D2}, s_{D3}, s_{D4}\}$

$E = \{e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$

$O = \{o_{D1}, o_{D2}, o_{D3}\}$

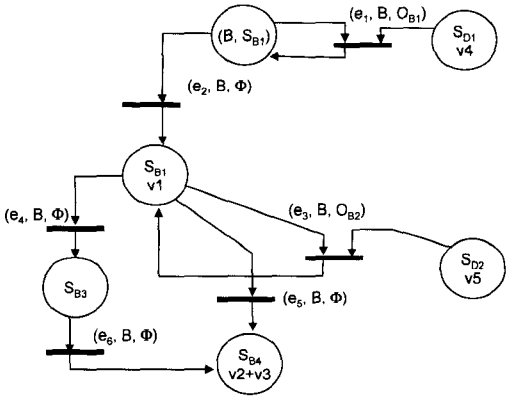
$T = \{(e_7, s_{D1}, s_{D2}), (e_9, s_{D2}, s_{D3}), (e_{10}, s_{D2}, s_{D4})\}$

$X = \{(e_8, o_{D1}), (e_{11}, o_{D2}), (e_{12}, o_{D3})\}$

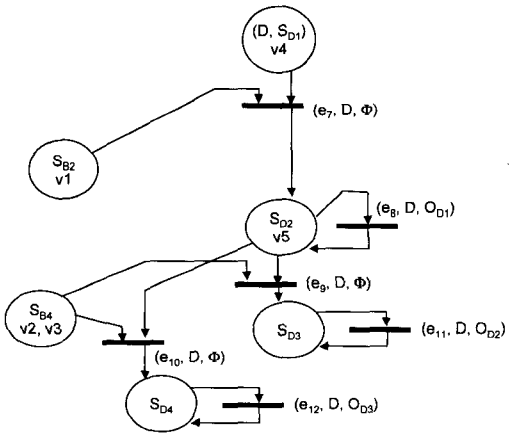
$V = \{v4, v5\}$

v4, v5 : 객체 B의 이벤트 e_1, e_3 발생을 위한 전달인자

위에서 제시한 가정과 3.2.2에서 제안한 평탄화된 상태차트 다이어그램 CPN으로의 전환 규칙과 알고리즘에 의하여 객체 B와 D의 상태차트 다이어그램을 CPN으로 전환하면 각각 (그림 13), (그림 14)와 같다.



(그림 13) 객체 B의 CPN



(그림 14) 객체 D의 CPN

객체들간의 상호작용은 메시지 전달에 의하여 이루어진다. 객체들간의 상호작용을 테스트하기 위하여 형성된 CPN들을 결합하여 단일 CPN을 구성한다. 단일 CPN 구성에서 객체들간의 상호작용에 의한 메시지 입/출력에 근거하여 CPN

들을 결합한다.

위에서 제시한 객체 B의 이벤트 e_1과 e_3의 발생은 각각 객체 D의 s_{D1}과 s_{D2}에서의 메시지 전달 v4, v5에 의해 점화하고, 객체 D의 이벤트 e_7, e_9, e_{10}의 발생은 각각 객체 B의 s_{B2}와 s_{B4}에서 메시지 전달 v1, v2, v3에 의해 점화한다고 가정한 객체 B와 D의 상호작용에 근거하여 객체 B와 D의 단일 CPN을 구성한다. 객체 C와 D의 단일 CPN도 같은 방법으로 구성할 수 있다. (그림 15)는 객체 B와 객체 D의 메시지 전달에 의한 상호작용에 근거하여 단일 CPN으로 구성한 것이다.

4. 테스트 케이스 생성 및 결과 분석

4.1 실험 환경

이 실험에서는 울트라 스팍 4 기반의 솔라리스 2.7에서 Design/CPN을 실행한다. Design/CPN으로 실행하면 테스트 케이스의 기반이 되는 도달성 그래프와 디스크립트를 추출할 수 있다. 도달성 그래프는 초기 플레이스로부터 점화하여 마지막 상태 플레이스로 이동하는 경로를 나타낸 트리이고, 디스크립트는 CPN의 플레이스에서 대응하는 도달성 그래프의 각 노드에 대해 토큰의 변화를 나타낸 것이다. 디스크립트를 이용하여 도달성 그래프에 노드이름과 에지 라벨을 붙이게 되면 서부투어를 얻을 수 있다.

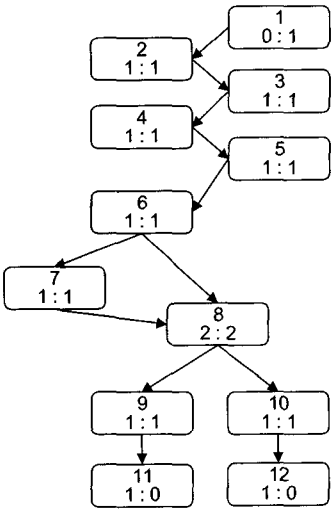
또한 Design/CPN의 시뮬레이터를 이용하여 실험이 제대로 진행되는지 상태를 알아본다. 시뮬레이터는 실행 시 "Fair Interactive," "Fair Automatic," "Fast Automatic"등 세가지 형태의 시뮬레이션을 제공한다.

"Fair Interactive"는 디버깅을 위해 사용되고 실행 속도가 가장 느리다. 이것은 브레이크포인터(breakpoint)라는 특정한 지점에서 실행을 멈출 수 있는 기능을 제공한다. 실행이 멈추었을 때, 바인딩이나 마킹의 수정과 같은 여러 실험이 수행될 수 있다. "Fair Automatic"은 브레이크포인터를 발생시키지는 않으나, 모델링의 정확성을 기하기 위하여 실행시간이 느린 편이다. "Fast Automatic"은 시행시간이 가장 짧으며, 디버깅이 불편한 단점이 있어 전체 시스템에 대한 일괄적인 시뮬레이션을 위하여 사용한다.

4.2 테스트 케이스 생성

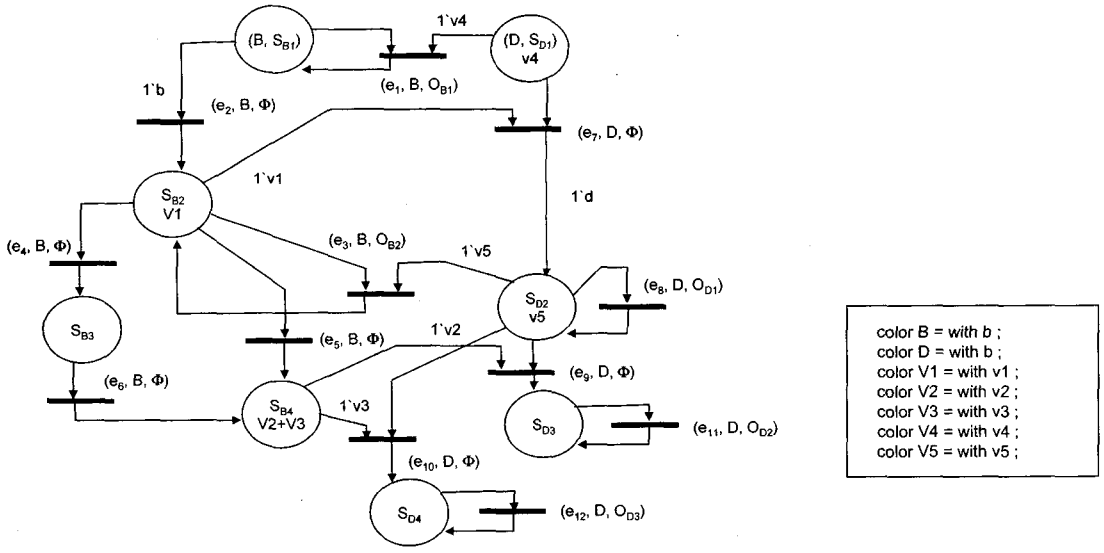
테스트 케이스를 생성하기 위하여 Design/CPN 편집기를 이용하여 (그림 15)의 CPN 표현을 입력한다.

시뮬레이션이 정확히 끝난 후, Design/CPN의 OG 생성 툴을 이용하여 (그림 16)과 같은 도달성 그래프를 얻는다.



(그림 16) 객체 B와 D의 도달성 그래프

도달성 그래프는 객체 토큰들이 CPN의 초기 상태 플레이스로부터 시작하여 마지막 상태 플레이스로 이동하는 경



(그림 15) 객체 B와 D 상호작용의 단일 CPN

로들을 트리 형태로 표현한 것으로서 노드번호, 부노드개수, 자노드개수로 구성된다. 도달성 그래프에 도달성 그래프에 대한 디스크립터인 (그림 17)을 이용하여 도달성 그래프의 각 노드와 노드간의 에지에 라벨을 붙여 서부투어를 생성하게 된다. 서부투어는 각 노드에 대한 상태 플레이스를 매핑시켜 상태의 흐름을 나타낸다. 예를 들면 (그림 16)에서 1-2-3-4-5-6-8-9-11로 이어지는 경로는 (그림 15)의 S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B4}-S_{D3}-S_{D3}의 경로에 해당하는데 이러한 경로를 서브투어라고 한다.

디스크립트는 CPN에서 각 플레이스가 점화함에 따른 토큰의 변화를 나타낸 것이다. 따라서 각 노드에 대해서 토큰을 비교하면 노드에서 자노드로 점화하기 위하여 어떤 토큰을 사용하였는가를 알 수 있다. (그림 17)은 노드 1에서부터 12까지의 디스크립트를 보여주고 있다.

1 New`Sb1 1: 1`b New`Sb2 1: 1`v1 New`Sb3 1: empty New`Sb4 1: 1`v2+1`v3 New`Sd1 1: 1`v4+1`d New`Sd2 1: 1`v5 New`Sd3 1: empty New`Sd4 1: empty	2 New`Sb1 1: 1`b+1`v4 New`Sb2 1: 1`v1 New`Sb3 1: empty New`Sab41: 1`v2+1`v3 New`Sd1 1: 1`d New`Sd2 1: 1`v5 New`Sd3 1: empty New`Sd4 1: empty	3 New`Sb1 1: empty New`Sb2 1: 1`v1+1`b New`Sb3 1: empty New`Sab41: 1`v2+1`v3 New`Sd1 1: 1`d New`Sd2 1: 1`v5 New`Sd3 1: empty New`Sd4 1: empty	4 New`Sb1 1: empty New`Sb2 1: 1`b New`Sb3 1: empty New`Sb4 1: 1`v2+1`v3 New`Sd1 1: empty New`Sd2 1: 1`v5+1`d+1`v1 New`Sd3 1: empty New`Sd4 1: empty
5 New`Sb1 1: empty New`Sb2 1: 1`b New`Sb3 1: empty New`Sb4 1: 1`v2+1`v3 New`Sd1 1: empty New`Sd2 1: 1`v5+1`d New`Sd3 1: empty New`Sd4 1: empty	6 New`Sb1 1: empty New`Sb2 1: 1`b+1`v5 New`Sb3 1: empty New`Sb4 1: 1`v2+1`v3 New`Sd1 1: empty New`Sd2 1: 1`d New`Sd3 1: empty New`Sd4 1: empty	7 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: 1`b New`Sb4 1: 1`v2+1`v3 New`Sd1 1: empty New`Sd2 1: 1`d New`Sd3 1: empty New`Sd4 1: empty	8 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: empty New`Sb4 1: 1`b+1`v2+1`v3 New`Sd1 1: empty New`Sd2 1: 1`d New`Sd3 1: empty New`Sd4 1: empty
9 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: empty New`Sb4 1: 1`b+1`v3 New`Sd1 1: empty New`Sd2 1: empty New`Sd3 1: 1`d+1`v2 New`Sd4 1: empty	10 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: empty New`Sb4 1: 1`b+1`v2 New`Sd1 1: empty New`Sd2 1: empty New`Sd3 1: empty New`Sd4 1: 1`d+1`v3	11 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: empty New`Sb4 1: 1`b+1`v3 New`Sd1 1: empty New`Sd2 1: empty New`Sd3 1: 1`d New`Sd4 1: empty	12 New`Sb1 1: empty New`Sb2 1: empty New`Sb3 1: empty New`Sb4 1: 1`b+1`v2 New`Sd1 1: empty New`Sd2 1: empty New`Sd3 1: empty New`Sd4 1: 1`d

(그림 17) 그림 16의 디스크립트

서부투어에 근거하여 이벤트들의 연속을 구한다. 예를 들면 서부투어 <S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B4}-S_{D3}-S_{D3}>의 경로에 해당하는 이벤트의 연속은 <e_1, e_2, e_7, e_8, e_3, e_5, e_9, e_{11}>이다. 생성된 이벤트 연속 <e_1, e_2, e_7, e_8, e_3, e_5, e_9, e_{11}>에 (그림 6)의 클래스 다이어그램에서 명세한 메소드들을 대응시켜 메소드의 연속인 테스트 케이스(1(), 2(), 7(), 8(), 3(), 5(), 9(), 11())를 얻는다. 같은 방법으로 서부투어의 다른 경로에 의한 이벤트 연속들을 구하고 메소드들을 대응시켜 테스트 케이스들을 구한다.

같은 방법으로 객체 C와 객체 D의 상호작용 테스트를 위한 테스트 케이스를 구할 수 있다. 상속성, 다형성 등 특성에 의한 비결정적인 문제는 메시지 전달에 의한 전달인자 V의 토큰속성 제어를 통하여 해결한다.

<표 1>은 객체 B와 객체 D 상호작용 테스트를 위해 생성된 서부투어와 메소드들의 연속인 테스트 케이스를 나타낸 것이다.

〈표 1〉 서부투어와 테스트 케이스

	서 부 투 어	테스트 케이스
1	S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B4} S_{D3}-S_{D3}	1( ), 2( ), 7( ), 8( ), 3( ), 5( ), 9( ), 11( )
2	S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B3} S_{B4}-S_{D3}-S_{D3}	1( ), 2( ), 7( ), 8( ), 3( ), 4( ), 9( ), 11( )
3	S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B4} S_{D4}-S_{D4}	1( ), 2( ), 7( ), 8( ), 3( ), 5( ), 10( ), 12( )
4	S_{D1}-S_{B1}-S_{B2}-S_{D2}-S_{D2}-S_{B2}-S_{B3} S_{B4}-S_{D4}-S_{D4}	1( ), 2( ), 7( ), 8( ), 3( ), 4( ), 10( ), 12( )

4.3 결과 분석

실험에서는 객체지향 소프트웨어의 평탄화된 상태차트 다이어그램들을 CPN으로 전환하고 Design/CPN 툴에 적용하여 도달성 그래프와 디스크립트를 생성한 후, 이에 대한 분석을 통하여 <표 1>과 같은 테스트 케이스를 자동 생성하였다.

테스트 케이스 길이 면에서는 일반적인 방법과 차이가 없으나 상호작용하는 객체들의 동적 테스팅을 위한 테스트 케이스가 생성되었고 실행에서 메시지 전달에 의한 전달인자의 토큰 속성을 부여하여 상속성과 다형성에서 비결정적인 문제를 해결하였으며 다음 상태 플레이스로 진행하도록 토큰을 주어 실행 불가능 경로를 제거함으로서 데트락과 같은 문제를 해결하였고 입출력 전달인자 토큰들로 파라메타들을 표시함으로서 제어흐름과 데이터 흐름 모두를 고려한 테스트 케이스를 생성하였다.

5. 결 론

객체지향 소프트웨어의 특성들은 소프트웨어를 설계하고 개발하는데 많은 잇점을 주는 것은 사실이지만 상속성, 다형성 등 특성들은 테스팅에 어려움을 증가시키고 있다. 객체의병행성, 동적바인딩, 상호작용 등 동적인 면들은 프로그램의 실행 시간에 한 멤버 함수의 호출이 여러 멤버 함수에 바인딩 될 수 있다는 것을 의미한다. 그러므로 상호작용하는 객체들의 동적 행위를 테스트하기 위한 연구가 필요하다.

이 논문에서는 객체지향 소프트웨어의 클래스 다이어그램과 상태차트 다이어그램들을 상속성과 다형성 등 특성을 고려하여 평탄화된 상태 차트 다이어그램으로 확장하였고 이를 CPN으로의 전환 규칙에 의하여 단일 CPN으로 구성하였으며 Design/CPN 툴에 적용하여 도달성 그래프와 디스크립트를 생성하였다. 도달성 그래프와 디스크립트 분석을 통하여 상호작용 하는 객체들을 테스트하기 위한 테스트 케이스를 생성하였다.

Design/CPN 툴을 이용하여 상호 작용하는 객체들의 동적행위 테스팅을 위한 테스트 케이스를 자동 생성하였고 실행에서 메시지 전달에 의한 전달인자의 토큰 속성을 부여하여 상속성과 다형성에 의한 비결정적인 문제를 해결하

였으며 다음 상태 플레이스로 진행하도록 토큰을 주어 실행 불가능 경로를 제거함으로서 데트락과 같은 문제를 해결하였고 입출력 전달인자 토큰들로 파라메타들을 표시함으로서 제어흐름과 데이터 흐름 모두를 고려한 테스트 케이스를 생성하였다.

향후 객체지향 소프트웨어 명세를 직접 CPN으로 전환하여 설계 및 테스트하는 방법의 연구와 Design/CPN 툴을 보다 편리하게 사용할 수 있는 연구가 필요하다.

참 고 문 헌

[1] Perry, D.E. and Kaiser, G.E., "Adequate Testing and Object-Oriented Programming," Journal of Object-Oriented Programming, pp.13-19, 1990.

[2] Rothernel, G. and Harrold, M.J., "Selecting Regression Tests for Object-Oriented Software," Technical Report pp.94-104, Clemson Univ., SC, March, 1994.

[3] Peterson, J.L., "Petri Net Theory and the Modeling of system," Englewood Cliffs, New Jersey, Prentice Hall Inc, 1981.

[4] Jensen, K., "COLOURED PETRI NETS-Basic Concepts, Analysis Methods and Practical Use-Volume 1-3," Stringer-Verlag, 1992, 1994, 1997.

[5] "Design/CPN Reference Manual for X-Windows Version 2.0," Meta Software Corporation, 1993.

[6] Watanabe, H., Tokuoka, H., Wu, W., Saeki, M., "A technique for analysing and testing object-oriented software using coloured Petri nets," In Asia Pacific Software Engineering Conference, IEEE Computer Society Press, pp.182-190, 1998.

[7] 김영집, 객체지향 소프트웨어의 상태기반 테스팅 방법, 박사학위논문, 충북대학교 대학원, 2001

[8] Harrold, M.J., McGregor, J.D. and Fitzpatrick, K.J., "Incremental Testing of Object-Oriented Class Structures," In 14th International Conference on Software Engineering, ACM, pp.201-208, 1992.

[9] Hong, H.S., Kwon, Y.R. and Cha, S.D. "Testing of object-oriented programsbased on finite state machines," In Asia Pacific Software Engineering Conference, IEEE Computer Society Press, Los Alamitos, California, pp.234-241, 1995.

[10] Robert, V. Binder, "The FREE-flow Graph : Implementation based Testing of Object Using State determined Flow," In Proceedings, 8th Annual Software Quality Week. San Francisco : Software Research, May, 1995.

[11] 이현정, 우성희, 오병호, 이상호 "실행불가능 경로가 제거된 테스트 케이스 생성", 정보과학회 춘계발표 논문집, 제25권 제1호, pp.443-445, 1998.

[12] Booch, G., Rumbaugh, J. and Jacobson, I., "The Unified Modeling Language User Guide," Addison-Wesley, 1999.

[13] Robert, V. Binder, "Testing Object-Oriented systems," Addison-Wesley, 2000.

이 인 혁

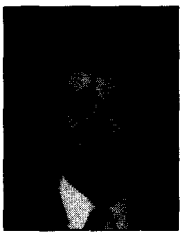
e-mail : leerg@selab.chungbuk.ac.kr

1992년 중국하얼빈공업대학 자동차설계학과(학사)

1998년 충북대학교 전자계산학과(이학석사)

1999년~현재 충북대학교 전자계산학과 박사과정

관심분야 : 소프트웨어 테스팅, 소프트웨어공학, 소프트웨어 재사용



구 연 설

e-mail : yskoo@cbucc.chungbuk.ac.kr

1964년 청주대학교 상학과

1975년 성균관대학교 경영대학원 전자자료처리학과(경영학석사)

1981년 동국대학교대학원 통계학과(이학석사)

1988년 광운대학교대학원 전자계산학과(이학박사)

1979년~현재 충북대학교 전기전자컴퓨터공학부 교수

충북대학교 전산소장, 자연과학대학장, 한국정보과학회 이사, 전산교육연구회회장, 충청지부장, 부회장 역임

관심분야 : 소프트웨어공학, 소프트웨어 테스팅, 알고리즘