ISSN(Print): 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.1.019

Derivation of State Transition Diagram from Class Using Tree Structure

Choi Soo Kyung† · Park Young Bom††

ABSTRACT

To improve the reliability and quality of software system, many studies of the testing based on state-transition diagram have been in progress. Existing studies tried to solve the complexity problem of state-transition diagram. But the development of test case demands the better way to derive and manage the state diagram with low complexity. In this paper, the STMT(State-Transition Mapping Tree) is proposed to decrease the complexity of state diagram without changing or loosing the original state or transition information. Comparing with other methods, the proposed method turns out to be less complex.

Keywords : Software Testing, State Diagram

트리 형태를 이용한 클래스의 단계별 상태 다이어그램 도출 기법에 대한 연구

최 수 경† · 박 용 범††

요 약

소프트웨어 시스템의 안정성 및 품질을 향상시키기 위해서 상태 다이어그램을 기반으로 한 테스트 기법들과 상태 다이어그램 도출에 관한 연구들이 진행되고 있다. 기존의 연구들은 일부 상태와 전이를 변경하여 상태 다이어그램의 복잡도 문제를 해결하고 있다. 그러나 테스트 케이스 도출에는 상태와 전이를 유지하면서 상태 다이어그램을 도출하는 방법이 필요하며 복잡도를 낮추는 방법도 필요하다. 본 논문에서는 상태나 전이의 변경 없이 복잡도를 감소시키기 위해 STMT(State-Transition Mapping Tree) 방법을 제안한다. 제안한 방법은 기존의 표기법과 비교하였을 때 복잡도가 낮다는 결과를 얻을 수 있었으며 시스템이 복잡해질수록 기존 방식에 비해 복잡도 개선에서 이점을 보였다.

키워드 : 소프트웨어 테스팅, 상태 다이어그램

1. 서 론

개발된 소프트웨어 시스템의 안정성 및 품질을 향상시키기 위해서는 소프트웨어의 테스트 과정이 필요하다. 일반적인 소프트웨어 테스팅은 소프트웨어 시스템 내의 결함을 검출하는 작업을 말하며 프로그램을 실행시켰을 때 결과가 명세서에 기술된 대로 산출되는가를 조사한다[1].

소프트웨어 테스트는 설계 단계부터 철저한 검증을 통해 안전성을 확보하기 위한 모델 기반 혹은 정형명세 기반의 테스트와 관련된 연구들이 활발히 진행되어 오고 있다[2]. 최근에는 상태 다이어그램을 기반으로 한 테스트 기법들이 연구되고 있는데 이러한 상태 다이어그램 기반 연구들은 커버리지를 이용하여 전이를 측정하거나 유·무효한 테스트 케이스를 생성하여 소프트웨어 시스템을 테스트 하는 방법들로 개발된 소프트웨어 시스템의 안정성을 테스트하는 방식으로 연구되고 있다[3].

상태 다이어그램은 테스팅이나 정형검증 등의 다양한 분야에서 유용하게 사용할 수 있는 동적 모델로써 많은 분야에서 사용되고 있다. 상태 다이어그램을 이용하여 테스트 케이스를 도출하거나 커버리지를 설정하여 상태 다이어그램의 전이를 측정하기 위해서는 개발된 소프트웨어 시스템으로부터 상태 다이어그램을 도출해내는 방법이 필요하다. 상태 다이어그램의 종류에는 전통적인 유한상태기계, D.Harel의 Statechart등 여러 가지가 있으며 본 논문에서는 UML의 State Diagram을 기반으로 하였다. 또한 UML의 State Diagram에서 제공하는 Construct는 STMT와 중복이 있어 제외 하였다. 상태 다이어그램을 도출하기 위해서는 클래스로부터 효과적으로 상태와 전이를 도출하는 방법이 필요하며 소프트웨어 시스템의 규모

※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2012-(H0301-12-3004)).
† 준 회 원 : 단국대학교 전자계산학과 석사과정
†† 종신회원 : 단국대학교 컴퓨터과학과 교수
논문접수 : 2012년 10월 5일
수 정 일 : 1차 2012년 11월 8일
심사완료 : 2012년 11월 27일
* Corresponding Author : Park Young Bom(ybpark@dankook.ac.kr)

에 따라 매우 복잡해 질 수 있다는 문제점 때문에 상태 다이어그램을 어디까지 보여줄 것인가에 대한 고려가 필요하다. 상태 다이어그램을 어디까지 보여줄 것인가에 대한 문제는 상태 다이어그램에 트리 구조의 깊이(Depth) 개념을 적용하여 깊이에 대한 문제로 나타낼 수 있다.

기존의 상태 다이어그램 도출에 관한 연구들은 클래스의 메서드 유형을 파악하여 일부 상태와 전이를 변경함으로써 상태 다이어그램의 복잡도 문제를 해결해나가고 있다. 하지만 상태나 전이를 변경시켜서 상태 다이어그램의 복잡도를 줄이는 방법은 소프트웨어 자동 분석 연구에 더 적합하며 소프트웨어 테스팅의 테스트 케이스 도출에는 변경된 상태와 전이로 인해 부적절한 테스트 케이스가 추가적으로 생성되는 문제점이 발생할 수 있다. 따라서 기존의 연구들과는 다르게 상태와 전이를 유지하면서 도출된 상태 다이어그램을 간소화하는 방법이 필요하다.

본 논문에서는 전이 변경 없이 도출된 상태 다이어그램의 복잡도를 감소시키기 위하여 생성된 상태 다이어그램을 본 논문에서 정의한 STMT(State- Transition Mapping Tree)라는 새로운 표기법을 적용하여 표기하는 방법을 제안한다. 또한 JAVA AST(Abstract Syntax Tree, 이하 AST)를 이용하여 소스 코드로부터 상태와 전이를 생성한 뒤, 클래스의 단계별 상태 다이어그램을 도출하는 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구들을 살펴보고 3절에서는 상태 다이어그램을 도출하기 위한 기법에 대해 설명한다. 4절에서는 지하철 개찰구 예제를 이용하여 제안한 단계별 상태 다이어그램 도출 기법을 적용한 예를 보인다. 마지막으로 5절에서는 결론 및 향후 연구 방향을 기술한다.

2. 관련 연구

2.1 기존 상태 다이어그램 도출 기법

상태 다이어그램 도출에 관한 연구에는 시나리오 기반과 비 시나리오 기반의 연구가 있다. 기존의 상태 다이어그램 도출에 관한 연구들은 시나리오로부터 동적 모델을 도출하는 연구들이 많이 진행되었다[4]. S.Uchitel과 J.Kramer의 연구에서는 UML의 시퀀스 다이어그램으로부터 상태 다이어그램을 생성하는 연구가 진행되었다[5]. S.Uchitel과 J.Kramer는 요구사항으로부터 각각의 시퀀스 다이어그램을 도출시킨 후에 시퀀스 다이어그램에 상태 다이어그램에서의 시작점과 종료점을 추가시키면서 시퀀스 다이어그램을 변형시켰다. 이 후 변형된 시퀀스 다이어그램을 하나의 순서로 만들면서 상태 다이어그램을 생성하였다.

J. Wittle, J. Schumann과 R. Kwan의 연구에서는 시나리오 집합으로부터 클래스의 상태 다이어그램을 생성하는 연구가 진행되었다[6,7]. J. Wittle, J. Schumann과 R. Kwan은 요구사항으로부터 시나리오 별로 시퀀스 다이어그램을 도출시켰다. 이 후 클래스를 도메인 지식형태로 표현하여 구성하였으며 상태 벡터라는 개념을 적용하여 각각의 시퀀스 다이어그램에 대입시켰다. 상태 벡터를 시퀀스 다이어그램에 적용함으로써 각각의 시퀀스 다이어그램이 상태-전이에 대한 정보를 가지게 되며 해당 정보를 바탕으로 상태 다이어그램을 도출한다. 하지만 앞서 언급한 시나리오 기반의 상태 다이어그램 도출 기법들은 시퀀스 다이어그램으로부터 상태 다이어그램을 도출해 내기 때문에 시퀀스 다이어그램을 필요로 하며 시퀀스 다이어그램을 도출시키기 위한 적절한 시나리오를 요구한다.

비 시나리오 기반의 상태 다이어그램을 도출해내는 연구들은 상태 다이어그램의 상태 전이의 중복 형태 등을 고려하여 상태와 전이를 변경하는 형식으로 진행되고 있으며 중첩된 전이를 나눠서 새로운 상태 전이를 생성해 내는 방식으로 상태 다이어그램의 복잡도를 줄여나가고 있다[9]. 비 시나리오 기반의 상태 다이어그램 도출 기법들은 시나리오 기반과 같이 시퀀스 다이어그램을 필요로 하거나 특정 시나리오를 필요로 하지 않는다. 일반적인 테스팅을 위해서는 소스코드로부터 상태 다이어그램을 추출하는 비 시나리오 기반의 상태 다이어그램 도출 방법이 적절하다.

2.2 AST(Abstract Syntax Tree)

추상 구문 트리(Abstract Syntax Tree, 이하 AST)는 소스 코드의 추상화된 구문의 트리 형태의 표현이다. 일반적인 관점에서 추상화 되었다는 표현은 실제 구문의 자세한 부분을 표현하지 않는다는 것을 말한다[8]. AST에서는 조건문의 분기가 하나의 노드로 표현된다. 예를 들어 소스코드 중에 If-else구문이 있으면 If 괄호 내용들과 else의 괄호 내용이 각각 “IfStatement”라는 표시로 하나의 노드로 표현이 된다.

이클립스는 JDT(Java Development Tool)라는 자바 개발 환경을 포함하고 있다. JDT는 자바개발과 관련하여 다양한 기능을 제공하기 위해 자바 파서와 AST를 내장하고 있으며 JDT에 내장된 자바 파서와 AST를 이용하면 소스 코드를 분석할 수 있다. 또한 AST는 자바 소스 코드에 대한 자세한 정보를 트리로 표현해 준다. 이때 자바 소스 코드는 ASTNode의 서브 클래스로 표현되며 ASTNode는 각 객체의 특정 정보를 제공해준다.

3. 상태 다이어그램 도출 및 표기

기존 방법 중 상태와 전이 변경 방법의 경우, 메서드의 유형에 따라 상태 전이를 변경하게 된다. 만약 메서드 내부 유형이 중첩 관계인 경우 각각의 세분화 하여 상태와 전이가 추가적으로 발생하게 된다. 또한 각각을 세분화하였기 때문에 경계 수가 증가하는데 이를 소프트웨어 테스팅 중 경계값 분석의 입장에서 부분적으로 봤을 때 변경되거나 추가된 전이와 상태로 인해 결함확률이 높아진 것을 확인할 수 있으며 실질적으로 코드 상에서는 구현된 부분과 다이어그램의 불일치로 적절하지 않은 테스트 케이스가 도출될 가능성이 발생한다. 원래는 구분되어 있지 않은 부분에 대해서 구분이 된 상태 다이어그램이 도출됨으로써 부적절한 테

스트 케이스가 추가로 도출될 가능성이 있다. 따라서 본 논문에서는 상태와 전이의 변경 없이 상태 다이어그램을 도출해내는 방법을 제안한다. 또한 기존의 상태 다이어그램 표기법의 경우 모든 상태들이 한 곳에 표현되어 있어서 시스템의 복잡도나 규모에 따라 상태 다이어그램의 복잡도가 증가한다는 단점이 있다. 또한 기존의 상태 다이어그램은 작성하는 방식에 따라서 다양한 형태로 표현될 수 있다[10]. 다양한 형태의 표현은 같은 시스템을 다르게 해석할 수 있게 하며 다른 복잡도를 가지게 할 수 있다. 따라서 본 논문에서는 한 곳에서 표현하는 상태 다이어그램의 표현 단위를 “맵(Map)”이라는 정의를 이용하여 한정하였으며, 맵 내부의 상태는 클래스의 메서드 단위까지만 도출하도록 정의하였다. 이 후 도출된 상태 다이어그램을 STMT로 표기하면 한 곳에서 표현되는 상태 다이어그램의 복잡도를 감소시킨다. STMT 표기법을 이용하면 상태 다이어그램을 트리 형태로 표기 할 수 있기 때문에 노드별로 상태들을 분리할 수 있다는 이점이 있다. 또한 상태 다이어그램을 클래스 단위인 맵으로 표기함으로써 여러 상태를 가지는 소프트웨어 시스템을 부분 별로 구분할 수 있다는 이점도 있다.

맵의 표기를 사용함으로써 소프트웨어 시스템의 상태 다이어그램에 대한 “깊이(Depth)”의 측정이 가능하다. 또한 각각의 깊이 별로 클래스에 대한 상태 다이어그램을 도출함으로써 소프트웨어 시스템의 복잡한 상태 다이어그램을 간소화 한다.

3.1 STMT 표기 형식 기본 정의

1) 상태와 맵

본 논문에서 새로 정의한 “상태(State)”와 “맵(Map)”은 클래스로부터 도출시킨 상태 다이어그램의 기반이 되는 것을 말하며, 다음과 같이 정의한다.

> [정의 1] 상태(State)는 하나의 클래스로부터 도출되며 트리거 이벤트에 따라 각각의 “다음 상태”로 전이된다.

> [정의 2] 맵(Map)은 하나 이상의 상태를 갖는 클래스를 말하며, 하나의 클래스에 대한 상태들은 맵으로 묶인다. 맵은 상태의 집합이다.

각각의 상태는 현재 상태(Current State)를 가지고 있으며, 다음 상태(Next State)로 넘어가기 위한 전이(Transition)인 트리거 이벤트(Trigger Event)와 동작(Action)을 가지고 있다. 또한 각각의 상태는 진입 동작(Entry Action)과 이탈 동작(Exit Action)을 가진다. 현재 상태는 내부의 동작에 따라서 다음 상태로 전이 된다. 이러한 상태 정의를 바탕으로 클래스에서 상태 다이어그램을 도출해 낼 수 있다. 클래스에서 생성자는 초기 시작 상태(Initial Pseudo State)의 진입 동작으로 도출되며, 각각의 메서드는 다음 상태로 가기 위한 이탈 동작으로 도출되거나 상태 다이어그램의 종료를 의미하는 종료 상태(Final State)로 가기 위한 이탈 동작으로 도출된다.

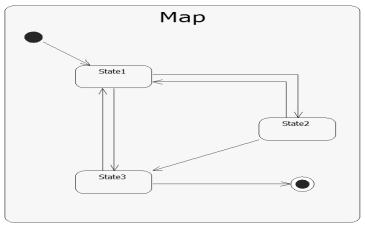

Fig. 1. State Diagram of Map(Internal)

하나의 클래스에 대한 상태들은 맵으로 묶이며 하나의 맵은 여러 상태를 갖는다. 맵의 내부 상태 다이어그램은 다음 Fig. 1과 같다.

Fig. 1의 표기는 STMT 표기에서 단말 노드인 맵을 확대시켰을 때 볼 수 있는 표기 방식이다.

클래스들의 연관 관계 설정 및 클래스 간의 상태 전이를 통해서 맵은 또 다른 맵 내부의 하위 상태가 될 수 있다. 또한 맵 간의 상태 전이는 상위의 맵인 “맵노드(Map Node, 이하 MN)”로 묶이게 되며 상위 맵인 MN이 존재할 경우 맵에 대한 깊이가 존재하게 된다. MN 내부의 맵 간의 상태 전이에서는 맵이 하나의 상태로 표기된다. 이 때 STMT에서 각각의 하위 맵은 깊이를 가지며 맵을 트리 형식으로 표기한다. STMT표기의 예는 다음 Fig. 2와 같다. STMT로 표기하였을 때 단말 노드인 맵은 깊이가 1로 설정된다. 여기서 단말 노드인 맵은 내부 상태에 또 다른 맵이 존재하지 않은 상태를 말한다. 깊이의 수치가 작을수록 클래스 내부의 상태에 가까우며 수치가 클수록 클래스들 간의 관계에 관련된 상태에 가깝다. 또한 최상위 루트 노드인 맵은 “맵 루트(Map Root, 이하 MR)”로 표기한다.

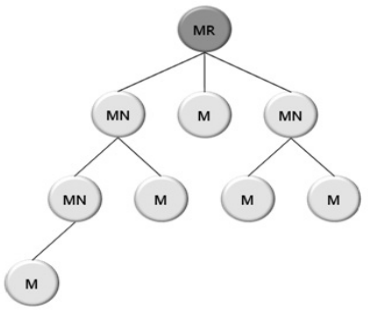

Fig. 2. Example of STMT notation

2) 맵 간의 관계

클래스로부터 상태 다이어그램을 도출한 뒤, 모든 상태를 관리하는데 어려움이 있으므로 본 논문에서는 “맵”이라는 새로운 정의를 추가하였다. 또한 맵으로 상태 다이어그램을 표기하면서 “맵 간의 관계”를 트리 형태로 표현이 가능해졌으며 상태 다이어그램을 트리화하였다. 또한 트리 형태를 이용하면 이클립스의 자바 구문 분석 프레임워크인 Java Development Tool에서 제공되는 JAVA AST를 이용하여 소스 코드로부터 상태와 전이를 자동으로 생성하는 자동화 도구와의 연동이 가능해진다.

본 논문에서 정의한 맵 간의 관계는 다음과 같다.

[정의 3]
M-M 관계
M-MN 관계(또는 M-MR)
MN-MN 관계(또는 MN-MR)

여기서 M은 맵을 의미하는 표기법이며, MN과 MR은 위치와 존재 수의 차이만 있을 뿐 내부적인 동작이나 구성은 같기 때문에 MR에 대한 관계 정의는 생략한다.

a) M-M 관계

M-M관계는 맵대 맵 관계이다. 예를 들어 맵(M1)과 맵(M2)의 M-M 관계는 Fig. 3과 같이 표기된다. 또한 M1과 M2에 대한 상태 다이어그램의 내부는 다음 Fig. 4와 같다.

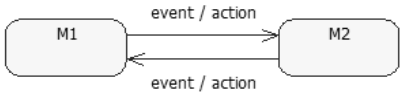

Fig. 3. State Diagram of M1, M2(Notation)

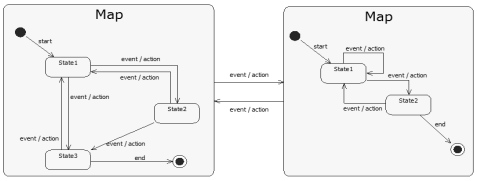

Fig. 4. State Diagram of M1, M2(Internal)

Fig. 4를 보면 Fig. 3의 표기에서 내부 상태가 상세화 되었다는 것 알 수 있다. 실제로 상태 다이어그램은 Fig. 4의 내부 모습들로 이루어진다. 이를 간소화하기 위해 Fig. 3의 표기를 사용함으로써 맵을 하나의 상태로 갖는 맵노드나 맵루트의 상태 다이어그램이 간소화됨을 알 수 있다. Fig. 3과 Fig. 4의 차이는 상위의 노드가 존재하느냐에 따라서 나눠진다.

M-M 관계에 있는 맵들의 상위에 맵노드나 맵루트가 존재하게 되면 M-M관계에 있는 맵들은 Fig. 3의 표기로 표현하게 되며 이표기는 맵노드나 맵루트 내부에 위치하게 된다.

b) M-MN 관계(또는 M-MR)

M-MN 관계는 맵 대 맵노드 관계이다. 예를 들어 맵(M1)과 맵노드(MN1)의 M-MN 관계는 다음 Fig. 5와 같이 표기된다. 또한 맵 대 맵노드 관계는 명칭만 다를 뿐 앞서 언급한 맵 대 맵루트(M-MR) 관계의 표기와 같다.

MN1과 M1에 대한 상태 다이어그램의 내부는 다음 Fig. 6과 같다.

Fig. 6을 보면 Fig. 5의 표기에서 내부 상태가 상세화 되었다는 것 알 수 있다. 실제로 상태 다이어그램은 Fig. 6의 내부 모습들로 이루어지며 더 상세하게는 Fig. 6의 맵노드 부분의 내부 상태들도({M1, M2}) 앞에서 설명했던 M-M관계의 내부 모습들로 이루어진다. 이를 간소화하기 위해 Fig. 5와 Fig. 3의 표기를 사용함으로써 맵을 하나의 상태로 갖는 맵노드나 맵루트의 상태 다이어그램이 간소화됨을 알 수 있다. Fig. 5와 Fig. 6의 차이는 Fig. 3과 Fig. 4의 차이와 마찬가지로 상위의 노드가 존재하느냐에 따라서 나눠진다. MN-M관계에 있는 맵들의 상위에 맵노드나 맵루트가 존재하게 되면 MN-M관계에 있는 맵들은 Fig. 5의 표기로 표현하게 되며 이표기는 맵노드나 맵루트 내부에 위치하게 된다.

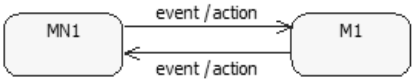

Fig. 5. State Diagram of MN1, M1(Notation)

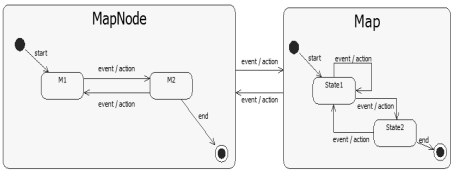

Fig. 6. State Diagram of MN1, M1(Internal)

c) MN-MN 관계(또는 MN-MR)

MN-MN 관계는 맵노드 대 맵노드 관계이다. 예를 들어 맵노드(MN1)과 맵노드(MN2)의 MN-MN 관계는 다음 Fig. 7과 같이 표기된다. 또한 맵노드 대 맵노드 관계는 명칭만 다를 뿐 앞서 언급한 맵노드 대 맵루트(MN-MR) 관계의 표기와 같다.

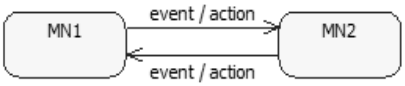

Fig. 7. State Diagram of MN1, MN2(Notation)

MN1과 MN2에 대한 상태 다이어그램의 내부는 다음 Fig. 8과 같다.

Fig. 8을 보면 Fig. 7의 표기에서 내부 상태가 상세화 되었다는 것 알 수 있다. 실제로 상태 다이어그램은 Fig. 8의 내부 모습들로 이루어지며 더 상세하게는 Fig. 8의 맵노드 부분의 내부 상태들도({M1, M2}, {M1, M2, M3}) 앞에서 설명했던 M-M관계의 내부 모습들로 이루어진다. 이를 간소화하기 위해 Fig. 7과 Fig. 3의 표기를 사용함으로써 맵을 하나의 상태로 갖는 맵노드나 맵루트의 상태 다이어그램이 간소화됨을 알 수 있다.

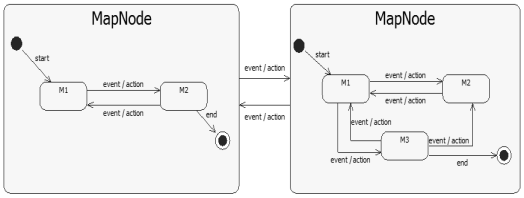

Fig. 8. State Diagram of MN1, MN2(Internal)

Fig. 7과 Fig. 8의 차이는 앞에서 설명한 관계들과 마찬가지로 상위의 노드가 존재하느냐에 따라서 나눠진다. MN-MN 관계에 있는 맵들의 상위에 맵노드나 맵루트가 존재하게 되면 MN-MN관계에 있는 맵들은 Fig. 7의 표기로 표현하게 되며 이표기는 맵노드나 맵루트 내부에 위치하게 된다.

3.2 단계별 상태 다이어그램 도출

본 논문에서 새로 정의한 "단계(Stage)"는 도출된 상태들을 맵으로 묶어서 트리화했을 때, 깊이에 의해 구분되는 것을 말하며, 구체적인 정의는 다음과 같다.

> [정의 4] 단계(Stage)는 상태 다이어그램에서 STMT가 도출되면 각각의 맵이 갖는 깊이에 의해 구분된다. 여기서 깊이가 1인 단말 노드는 단계의 값도 1이다.

상태 다이어그램에서 깊이는 한 상태와 그 상태를 포함하는 부모 상태와의 관계를 통해 측정할 수 있다. 더 이상 자식 상태를 가지지 않는 상태는 트리 구조에서의 단말 노드와 같으며 깊이는 1로 측정된다. 단계는 깊이에 의해 도출되며 단계의 수치가 적을수록 클래스 내부의 상태 다이어그램과 가깝다. 단계의 표기는 M(N)으로 표기하며 N은 단계의 번호이다. 예를 들어 [1단계]의 경우 최하위 단말 노드의 내부 상태 다이어그램의 도출을 말한다.

3.1절에서 Fig. 2를 예를 들면 각각의 "단말 노드"에 의해 단계가 다를 수 있다는 것을 알 수 있다. 제일 왼쪽 아래의 맵(M)의 경우, MR-MN-MN-M의 단계로 나눠져 있다는 것을 알 수 있다. 이 경우 MR[4단계]-MN[3단계]-MN[2단계]-M[1단계]로 볼 수 있으며 MR(4)-MN(3)-MN(2)-M(1)로 표기 한다. MR-M 단계로 나눠진 맵의 경우 MR[2단계]-M[1단계]로 볼 수 있으며, MR(2)-M(1)로 표기한다. 각각의 단계는 3.2절에서 설명한 맵 간의 관계 형식으로 표기된다. Fig. 2의 제일 왼쪽 아래의 맵(M)을 예를 들어 단계별 상태 다이어그램 표기하면 다음과 Fig. 9와 같다.

Fig. 9에서 도출된 상태 다이어그램은 단계별로 분류 할 수 있으며 각각 세분화 시킬 수 있다. 상태 다이어그램을 클래스의 포함여부에 따라 단계별로 구성하면 소프트웨어 시스템의 상태 다이어그램을 필요한 부분별로 볼 수 있게 해주거나 최상위 부분에서 큰 상태들의 동작을 볼 수 있다.

본 논문에서 제안하고자 하는 상태 다이어그램 도출 기법은 클래스로부터 STMT를 구성하기 위한 단계별로 상태 다이어그램을 도출하는 기법이다. 각 단계별 상태 다이어그램 도출 순서는 다음 Fig. 10과 같은 절차를 갖는다.

소프트웨어 시스템의 상태 다이어그램은 "어디까지 보여줄 것인가"를 고려하지 않으면 복잡한 상태 다이어그램을 도출할 가능성이 있다. 또한 복잡한 상태 다이어그램은 소프트웨어 시스템이 어떻게 작동하는지 명확하게 정의해주는 상태 다이어그램의 목적을 잃게 한다. 단계별로 구성된 상태 다이어그램은 어느 정도까지 그릴 것인지에 대한 깊이의 구분을 고려할 필요 없이 상태다이어그램을 도출 할 수 있

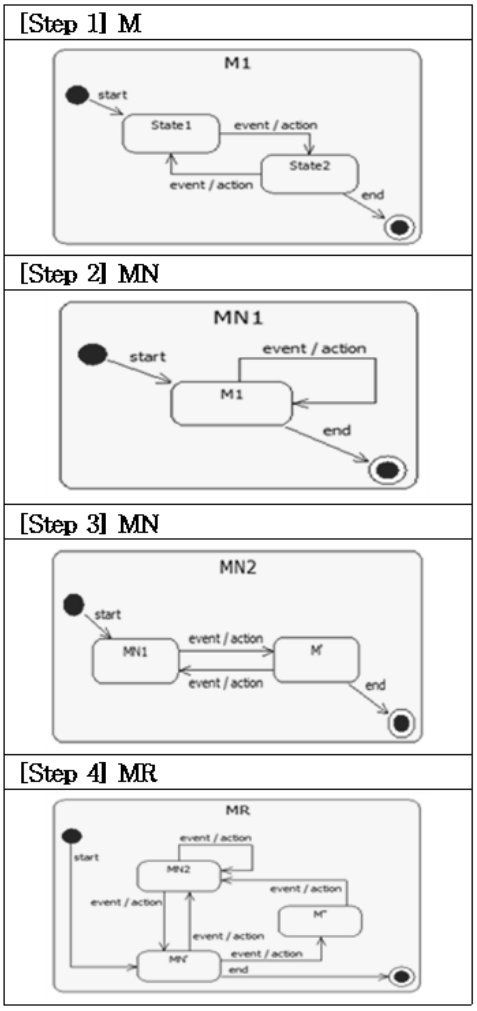

Fig. 9. Step by step notation of State Diagram

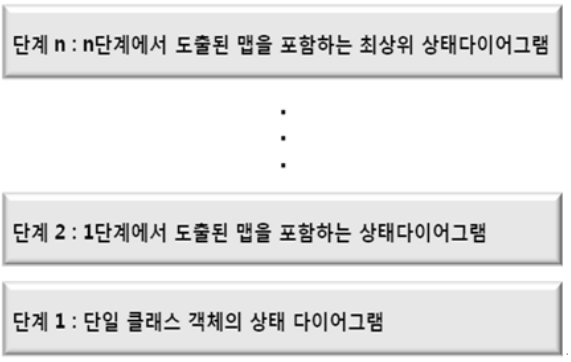

Fig. 10. deduction of state diagram step-by-step process

다.단계별 상태 다이어그램 도출 방법을 적용하기 위한 전체적인 프로세스는 다음 Fig. 11과 같다.

Fig. 11에 표시되어 있듯이 상태 다이어그램 도출 의 대략적인 과정은 다음과 같다. 먼저 소스 코드를 분석하여 AST를 생성한 후, 생성된 AST를 상태와 맵 단위로 재구성한다. 그 다음 재구성된 AST를 이용하여 3절에서 제안한 맵에 의한 트리 형식으로 표기한다. 마지막으로 각 단계별

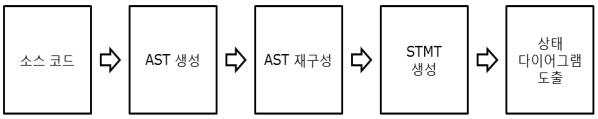

Fig. 11. Deduction of State Diagram Process

상태 다이어그램 도출 기법을 통해서 최종적인 상태 다이어그램을 도출시킨다.

이클립스의 자바 구문 분석 프레임 워크인 Java Development Tool에서 제공되는 JAVA AST를 이용하여 소스코드로부터 도출된 일반적인 AST 결과를 이용하면 맵의 개념을 넣어 AST를 재구성 할 수 있다. 재구성된 AST는 하나의 클래스에서 하나의 맵으로 설정할 때 기존에 도출된 AST 결과로부터 각 클래스의 각 노드는 맵으로 정의하며, 클래스 내부의 메서드 유형 정보가 담겨있는 속성으로부터 맵 내부의 상태와 정의를 정의한다. 새롭게 재구성된 AST는 XML 파일로 저장된다. XML 파일에는 맵 별로 기존 AST에서 도출된 클래스의 연결 정보를 이용하여 고유 ID를 가지며, 각각의 맵은 자신의 부모와 자식에 대한 맵 관계 정보를 갖으며 자신이 가지고 있는 상태와 전이에 대한 정보도 갖는다. 이렇게 재구성된 AST는 STMT 생성 단계에서 이용된다. 재구성된 AST의 맵 관계 정보를 이용하여 연관성 있는 맵 루트, 맵 노드, 맵을 도출하여 STMT를 구성한다. 구성된 STMT로부터 앞서 단계별 상태 다이어그램 도출에서 예로 들었던 Fig. 2처럼 MR(4)-MN(3)-MN(2)-M(1) 형태로 표기하게 된다. 표기된 형식의 각각의 단계는 단계별 상태 다이어그램 도출 방법대로 상태다이어그램을 도출시키는 프로세스를 따른다. 이 때 STMT에서도 쓰였던 재구성된 AST 정보를 이용하여 맵 내부의 상태와 전이에 대한 정보를 이용하여 각각의 단계에서 상태 다이어그램을 도출한다.

상태 다이어그램 도출 프로세스 중 소스 코드로부터 AST를 생성하는 과정은 자동화가 가능하다. JAVA AST를 이용하면 소스 코드로부터 원하는 형식으로 상태와 전이를 생성할 수 있다. 이 후 생성된 상태와 전이를 통해 맵에 의한 트리를 도출 시킨 후 상태 다이어그램을 도출시키는 자동화 도구에 대해 생각해 볼 수 있다.

4. 단계별 상태 다이어그램 도출 기법 적용

3절에서 제안한 기법에 따라 소프트웨어 시스템의 클래스로부터 상태 다이어그램을 도출해 낼 수 있다. 따라서 본 장에서는 3장에서 제안한 단계별 상태 다이어그램 도출 기법을 기반으로 상태 다이어그램을 도출하는 활용 사례를 보였다.

4.1 지하철 개찰구 예제

본 절에서는 지하철 개찰구(Turnstile) 예제를 이용하여3.1절에서 정의한 상태와 맵을 바탕으로 단계별 상태 다이어그램 도출 기법을 적용한 예를 보인다. 지하철 개찰구 예제는 Turnstile, State, Locked, Unlocked 클래스를 가지고 있다. 여기서 State는 가상 추상클래스이며, Locked와 Unlocked클래스는 State클래스를 상속받고 있다. Turnstile 클래스에서 Locked와 Unlocked를 생성하여 coin(동전 투입)과 pass(지나감)이라는 이벤트에 대한 전이를 일으켜서 다음 상태로 넘어가게 한다. 또한 Locked와 Unlocked 클래스는 내부 상태를 갖는다. Locked는 “Steady State(정상동작)”와 “Beep(경보울림)”라는 상태를 갖는다. Unlocked는 “Steady State(정상동작)”와 “returned(반환)” 상태를 갖는다.

지하철 개찰구 예제의 클래스의 수는 추상 클래스인 State를 제외하면 총 3개로 구성된다. 이 3개의 클래스에 3절에서 제안한 단계별 상태 다이어그램 도출 기법을 적용하여 상태 다이어그램을 도출한다. 단계별 작업을 하기 전에 지하철 개찰구 예제의 이해를 돕기 위해 상태 전이를 명확히 할 필요가 있다. 따라서 다음과 같이 지하철 개찰구에 대한 상태 전이 테이블을 작성하였다. 다음 Table 1은 지하철 개찰구 예제에 대한 상태 전이 테이블이다.

Table 1. State Transition Table of Turnstile

State	Event	Next State	Action
Locked	Coin	Unlocked	Unlock
Locked	Pass	Locked	Alarm
Unlocked	Coin	Unlocked	Return
Unlocked	Pass	Locked	Lock

지하철 개찰구의 상태 전이 테이블에서 Locked과 Unlocked은 각각의 클래스로 구성되어 있다. Locked과 Unlocked 클래스는 내부적인 상태 전이를 갖는다.

Locked에 대한 내부 상태 전이 테이블은 다음 Table 2와 같으며, Unlocked에 대한 내부 상태 전이 테이블은 다음 Table 3과 같다.

Table 2. State Transition Table of Locked

State	Event	Next State	Action
Steady State	Alarm	Beep	OnBeep
Beep	Coin	Steady State	Coin

Table 3. State Transition Table of Unlocked

State	Event	Next State	Action
Steady State	Coin_return	Returned	Return
Returned	Return completion	Steady State	-

도출된 상태 전이 테이블을 통해서 지하철 개찰구 예제의 Locked과 Unlocked 클래스가 최하위 “단말 노드”인 것을 유추 할 수 있다. 또한 Locked의 내부 상태들은 Locked이라는 맵에 묶이며, Unlocked의 내부 상태들도 Unlocked이라는 맵에 묶일 수 있다.

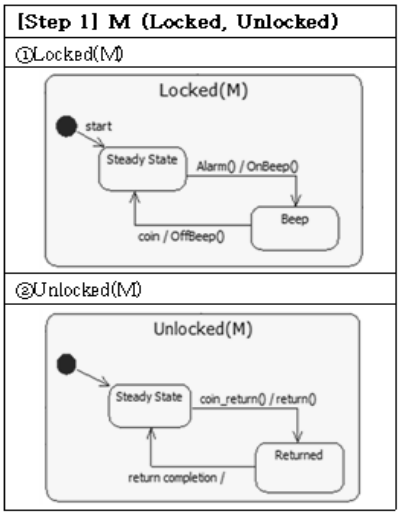

Fig. 12. Result of Step 1 - Turnstile

1단계의 단일 클래스의 상태 다이어그램을 도출한 결과는 Fig. 12와 같다. 이 지하철 개찰구 예제에서는 2단계 이후에 더 이상 도출될 맵이 없기 때문에 2단계의 결과가 최상위 루트 노드인 MR로 도출된다.

1단계에서 도출된 맵을 포함하는 상태 다이어그램을 도출하는 2단계 작업을 한 결과는 다음 Fig. 13과 같다.

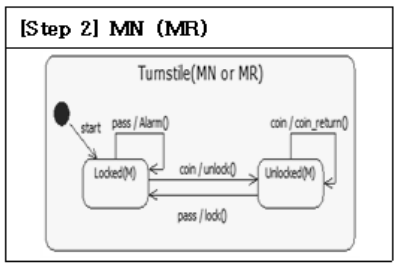

Fig. 13. Result of Step 2 - Turnstile

또한 추가적으로 Locked(M)과 Unlocked(M)을 M-M관계로 표현이 가능하다. Locked와 Unlocked로부터 도출한M-M 관계의 결과는 다음 Fig. 14와 같다.

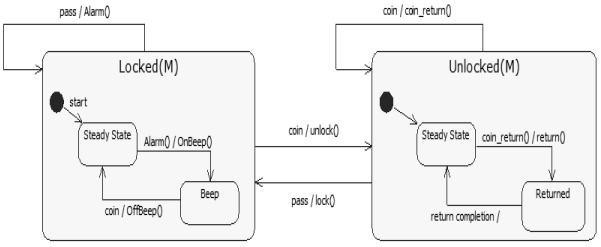

Fig. 14. Relation between M and M - Turnstile

각 단계별로 상태 다이어그램을 도출한 결과로부터 지하철 개찰구의 상태를 STMT로 표기한 예는 다음 Fig. 15와 같다.

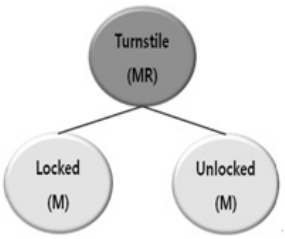

Fig. 15. STMT of Turnstile

결과적으로 제안한 표기법의 형식으로 지하철 개찰구 예제를 표현하면 Fig. 13과 같은 결과를 얻는다. Fig. 15의 경우 Fig. 13을 간단하게 도식화된 형태라고 볼 수 있다. 향후 테스트 케이스를 도출할 때에는 Fig. 15의 표기법이 사용되며 일반적인 상태 다이어그램 표기에는 Fig. 13의 표기 형태가 이용된다.

4.2 표기 형식 비교

본 절에서는 본 장에서 사용된 지하철 개찰구 예제로 기존의 상태 다이어그램 표기 형식과 제안한 상태 다이어그램 표기 형식을 비교한다.

본 논문에서 제안한 상태 다이어그램 표기 형식을 이용하면 기존의 상태 다이어그램 표기 형식보다 복잡도를 줄일 수 있다는 이점이 있다. 지하철 개찰구 예제로 두 표기 형식을 비교해보면 미시적 구조로 봤을 때는 복잡도가 같으나 거시적 구조로 봤을 때는 복잡도가 달라짐을 알 수 있다. 각각의 표기 형식으로 나타낸 결과는 다음 Fig. 16과 Fig. 17과 같다.

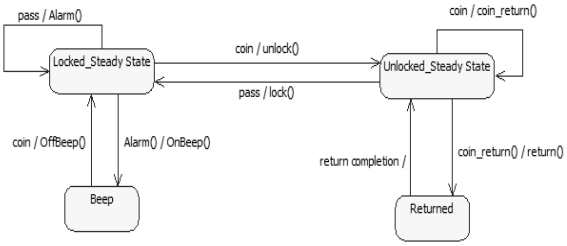

Fig. 16. Existing Notation of State Diagram

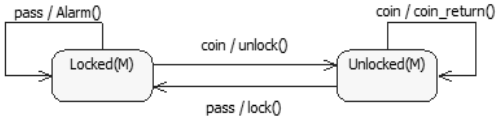

Fig. 17. Proposed Notation of State Diagram

두 표기 형식의 복잡도를 측정하기 위해 Derr가 제안한 상태 다이어그램 복잡도 메트릭을 이용하였다. Derr는 객체 모델링 기법을 사용하여 상태 다이어그램의 상태와 전이의 수로 상태 다이어그램의 복잡도를 측정하는 메트릭을 제안하였다[11]. 본 논문에서 복잡도 측정을 위해 사용한 Derr의

Table 4. Complexity Metric of State Diagram

	Metric Name	Description
Size	NSS	Number of single state
	NCS	Number of multi state
Complexity	NT	Number of transition
	CC	|NSS-NT|+2

Table 5. Result of Coplexity

	NSS	NCS	NT	CC
Previous	4	-	8	6
Proposed	2	-	4	4

메트릭을 요약한 표는 다음과 Table 4와 같다. 또한 Derr의 메트릭을 이용하여 기존 표기형식과 제안한 표기형식의 복잡도를 계산해보면 다음 Table 5와 같은 결과가 나온다.

위의 결과를 보면 기존 형식은 복잡도가 6이지만 제안한 형식은 복잡도가 4이다. 지하철 개찰구 예제의 경우 단순 상태 4개를 갖는 비교적 간단한 시스템이라 복잡도에 큰 차이는 보이지는 않지만 제안 형식이 기존 형식보다 복잡도가 낮다는 것을 알 수 있다. 또한 제안한 형식은 시스템이 복잡해질수록 상태 다이어그램 복잡도가 낮아지는 이점을 갖는다.

5. 결론 및 향후 연구 방향

본 논문에서는 클래스를 기반으로 상태 다이어그램을 구성하는 "상태"와 "맵"을 추출하고 상태의 집합인 맵을 기준으로 소프트웨어 시스템의 상태 다이어그램을 도출하는 기법을 제안하였다. 본 논문에서 제안한 상태 다이어그램 도출 기법의 의의는 다음과 같다.

첫째, 일반적인 상태 다이어그램은 깊이에 대한 고려가 없으면 상대적으로 복잡해질 수 있다. 클래스의 단계별 상태 다이어그램 도출 기법을 제시함으로써 깊이에 대한 고려 없이 상태 다이어그램을 간소화 시킬 수 있다.

둘째, 상태 다이어그램 도출 기법에 관한 연구들은 대부분이 상태 다이어그램의 복잡도를 클래스의 메서드 유형을 파악하여 전이를 변경하는 형식으로 진행되고 있다. 본 논문에서 제안한 상태 다이어그램 도출 기법은 기존의 연구들처럼 전이를 변경하면서 간소화 시키는 것이 아닌 기존 상태 전이를 기반으로 상태를 맵이라는 하나의 단위로 묶음으로써 해결하는 방법을 제안하였다.

본 논문에서는 트리의 형식과 개념을 이용하여 단일 클래스로부터 단계별로 상태 다이어그램을 도출하는 기법을 제안하였다. 도출하는 기법을 각 단계로 나누어 세분화 시켰으며 도출된 결과는 맵 별로 또는 부분별로 나눠서 볼 수 있게 하였다. 또한 맵으로 도출된 결과들은 STMT로 구성되어 도출될 수 있게 하였다. 본 논문에서 제안한 상태 다이어그램 도출 기법은 향후 AST를 이용한 상태 다이어그램 도출 자동화 도구에 대한 연구가 필요하며, 소프트웨어 시스템에서 상태 다이어그램을 도출해 낼 때, 클래스의 관계를 고려하여 상태 다이어그램을 도출해낼 수 있는 체계적인 방법에 대한 연구가 필요하다. 또한 STMT 표기를 이용한 테스트 케이스 추출 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] B.Beizer, "Software System Testing, 2nd. Ed," van Nostrand Reinhold, 1990.

[2] A. Pretschner, O. Slotosch, E.Aiglastorfer and S. Kriebel, "Model based testing for real," Software Tools for Technology Transfer, 5(2-3):140-157, March, 2004.

[3] IPL Information Processing Ltd, "Testing State Machine with AdaTEST and CANTATA," IPL paper, Mar., 2011.

[4] H. Liang, J. Dingel and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," In SCESM, pp.5-12, 2006.

[5] S. Uchitel and J. Kramer. "A workbench for synthesizing begaviour models from scenarios," In ICSE, pp.188-197, 2001.

[6] J. Whittle and J. Schumann, "Generating statechart designs from scenarios," In ICSE, pp.314-323, Jun., 2000.

[7] J. Whittle, R. Kwan and J. Saboo, "From scenarios to code: An air traffic control case study," In ICSE, pp.490-495, May, 2003.

[8] L. Kwang-Min, B. Jung Ho and C. Heung Seok, "An Automatic Construction Approach of State Diagram from Class Operations with Pre/Post Conditions," DOI: 10.3745/KIPSTD.2009.16-D.4.527, 2009.

[9] K. RyoungKwo, H. SeungAn and K. Gihwon, "Test Requirements for MC/DC Coverage," Software Engineering, 2010.

[10] R.V. Binder, "Testing Object-Oriented Systems: Models, Patterns, and Tools," Addison Wesley, 1999.

[11] K. Derr, "Applying OMT," SIGS Books, Prentice Hall, 1995.

최 수 경

e-mail : krsoogom@dankook.ac.kr

2011년 건양대학교 컴퓨터학과(학사)

2011년~현 재 단국대학교 전자계산학과 석사과정

관심분야 : Software Engineering, Software Testing

박 용 범

e-mail : ybpark@dankook.ac.kr

1991년 N.Y. Polytechnic University Science & Engineering(Ph.D.)

1993년~현 재 단국대학교 컴퓨터과학과 교수

관심분야 : Intelligent Software Engineering, Security Software