컴포넌트 기반 소프트웨어의 상호운용성 테스트를 위한 유스케이스 기반 테스트 케이스 생성 기법

(Test Case Generation Techniques based on Use Cases for Interoperability Test of Component-Based software)

유 철 중 † 노 혜 민 ††
(Cheol-Jung Yoo) (Hye-Min Noh)

요 약 사전 제조된 컴포넌트들을 재사용하여 애플리케이션을 개발하는 컴포넌트 사용자의 관심사는 사용자의 요구사항에 따라 적절히 다른 컴포넌트들과 협동하는지 여부를 확인하는 것이다. 따라서 컴포넌트 기반 소프트웨어의 경우 개발 환경이 아닌 새로운 환경에서 소프트웨어를 구성하는 컴포넌트들이 잘 연동되는지를 테스트할 수 있는 상호운용성 테스트에 관련된 연구가 중요시 되고 있다. 본 논문에서는 컴포넌트 기반 소프트웨어의 상호운용성 테스트를 위한 테스트 모델을 정의하고, 유스케이스 명세로부터 테스트 모델을 생성한 후 생성된 모델로부터 테스트 케이스를 생성하는 기법을 제안한다. 또한 테스트 모델로부터 테스트 시퀀스를 생성하는 프로시저를 구현한 도구를 소개한다.

키워드 : 상호운용성 테스트, CBD, 테스트 케이스 생성

Abstract The major concern of component users who develop applications using the existing components is to confirm whether a component is collaborating with the different components in accordance with the requirements. Therefore, interoperability testing whose role is to check whether components collaborate with each other within the new operating environment not within the component development context of each component is considered as an importance research topic. In this paper, we propose a test case generation technique for interoperability test of component based software. The proposed technique defines a test model for generating test cases. The proposed technique generates test models from the use case specification and thereafter from these models, test cases for interoperability testing are derived. In addition, we describe a tool which implements the procedures for generating test sequences from test models.

Key words : Interoperability Testing, CBD, Test Case Generation

1. 서 론

CBD(Component-Based Development)는 요구사항 분석, 설계, 구현, 테스트, 배치 및 기타 기술적인 지원 등을 포함하는 소프트웨어 개발 주기의 모든 측면 및 단계를 컴포넌트에 기반을 두고 있는 소프트웨어 개발 접근법이다[1]. CBD의 장점은 새로운 소프트웨어를 개발할 때 개발자 자신이 기존에 개발한 사전 제조된 컴포넌트들뿐만 아니라 전문 컴포넌트 벤더들이 제공한 고품질의 컴포넌트들을 구입하여 재사용할 수 있다는 점이다. 그러나 이러한 컴포넌트들은 새로운 애플리케이션 구축을 위해 개발 환경과는 전혀 다른 새로운 환경에 배치될 수 있다[2].

· 이 논문은 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-521-D00419)

† 정 회 원 : 전북대학교 응용시스템공학부 교수
cjyoo@chonbuk.ac.kr
†† 정 회 원 : 전북대학교 전자정보 BK사업단 기금교수
hmino@chonbuk.ac.kr

논문접수 : 2008년 8월 4일
심사완료 : 2009년 3월 4일

정보과학회논문지: 소프트웨어 및 응용 제36권 제5호(2009.5)

상호운용성은 전체 소프트웨어를 이루는 타 구성요소와 상호작용할 수 있는 소프트웨어 프로덕트의 능력을 나타내는 척도이며, 상호운용성 테스트는 소프트웨어 프로덕트의 상호운용성을 점검하기 위한 테스팅 과정이다[3]. 사전 제조된 컴포넌트들을 재사용하여 애플리케이션을 개발하는 컴포넌트 사용자의 관심사는 애플리케이션 구축 후 재사용된 컴포넌트가 새로운 프레임워크에 통합 가능하며, 사용자의 요구사항에 따라 적절히 다른 컴포넌트들과 협동하는지 여부를 확인 하는 것이다. 따라서 컴포넌트 기반 소프트웨어의 경우 개발 환경이 아닌 새로운 환경에서 소프트웨어를 구성하는 컴포넌트들이 잘 통합되어 연동되는지를 테스트할 수 있는 상호운용성 테스트에 관련된 연구가 중요시 되고 있다[4].

테스트 관점에서 최근 중요한 사항으로 등장하고 있는 것 중 하나는 테스트의 기반이 되는 모델에 관한 것이다[5]. 전통적인 테스트의 경우 자연어를 기반으로 하는 매우 추상화된 명세 또는 프로그램 소스코드를 이용한다. 자연어와 소스코드는 컴포넌트 기본 소프트웨어의 테스트 기반 모델로는 적절하지 못하다. 그 이유는 자연어는 모든 필요한 테스트 가공물을 유추하기에 충분할 만큼 정형적이지 못하며, 또한 소스코드는 컴포넌트 기반 개발을 고려한다면 컴포넌트 사용자는 구현 사항을 알 수 없기 때문에 실제 가치 있는 정보가 아니기 때문이다[6]. 최근에는 UML을 기반으로 여러 가지 다이어그램을 이용한 모델기반 테스팅(Model-Based Testing) 접근과 정형명세 및 유한상태머신(Finite State Machine)을 이용한 비 UML기반(Non-UML-Based)접근의 모델기반 테스팅이 여러 가지 소프트웨어 구축 패러다임과 도메인에 맞춰 제안되었고 점차 증가추세에 있다[7,8].

컴포넌트 기반 소프트웨어가 잘 만들어진 개개의 컴포넌트들이 특정 환경에서 조립 및 통합되어 구축된다는 관점에서 볼 때 상호운용성 테스트는 매우 중요한 의미를 지닌다. 그러나 현재 컴포넌트 기반 소프트웨어를 구성하는 컴포넌트들 사이의 상호운용성을 점검하는 체계적이며, 최적화된 상호운용성 테스트에 관한 연구가 테스트 모델을 기반으로 진행 중에 있다[8]. 또한 상호운용성 테스트를 위한 테스트 케이스 선정에 있어서 중복된 테스트 케이스로 인한 비용낭비와 완전하지 못한 테스트 케이스로 인하여 내재된 오류를 검출해내지 못하는 상황이 발생할 수 있다[8]. 그리고 수작업이나 잘 정의된 테스트 제어 절차 없이 임의로 이루어지는 테스트가 갖는 비효율성을 극복하기 위하여 테스트 케이스 생성 및 자동화가 필요한 실정이며, 이를 위해 체계적이고 최적화된 테스트 케이스 생성을 위한 상호운용성 테스트에 적합한 테스트 모델의 정의가 필요하다.

본 논문에서는 컴포넌트 기반 소프트웨어의 상호운용성 테스트에 필요한 테스트 모델 및 테스트 케이스 생성 기법을 제안하고자 한다. 본 논문의 연구 범위는 다음 세 가지로 요약할 수 있다.

- 상호운용성 테스트 케이스 생성을 위한 테스트 모델 제안
- 제안된 테스트 모델로부터 상호운용성 테스트 케이스 생성 기법 제안
- 제안된 상호운용성 테스트 케이스 생성 기법을 적용한 테스트 시퀀스 자동생성 도구 소개

본 논문은 다음과 같이 구성되어 있다. 제2장에서는 관련 연구로서 컴포넌트 기반 소프트웨어의 특성 및 테스트를 위해 고려해야 할 사항들을 소개한다. 제3장에서는 상호운용성 테스트 케이스 생성을 위한 테스트 모델을 정의하고 모델 생성 기법을 제안하며, 제4장에서는 제 3장에서 제안한 기법을 토대로 생성한 테스트 모델을 기초로 하여 테스트 케이스의 생성기법을 제안하고 적용 사례를 제시한다. 제5장에서는 적용한 사례에 대한 분석결과를 기술한다. 제6장에서는 본 논문의 결론과 제안한 기법의 의의에 대하여 기술한다.

2. 컴포넌트 기반 소프트웨어

2.1 컴포넌트 기반 소프트웨어 개발

컴포넌트 기반 소프트웨어 개발의 주요 아이디어는 모든 것을 새로 개발하지 않고 준비된 유용한 부분들을 재사용함으로써 새로운 소프트웨어 프로덕트를 구축하는 것이다[9]. 현재 제기되고 있는 컴포넌트 기반 소프트웨어 개발 이점들은 재사용을 통하여 전통적인 소프트웨어 개발 방법보다 더 높은 비용절감 효과를 얻을 수 있다는 가정을 기반으로 하고 있다[2]. 기존에 작성된 컴포넌트를 재사용한다는 측면에서 살펴보면 이러한 가정은 설득력이 있다. 그러나 전적으로 그렇다고 할 수는 없다. 이러한 가정은 이미 작성된 컴포넌트들을 새로운 환경에 적용하는 비용과 컴포넌트들을 새로운 환경에 통합시키는 비용이 전통적인 소프트웨어 개발 방법으로 프로덕트를 개발하는 비용보다 더 적을 경우에만 사실로서 인정할 수 있다[9].

소프트웨어 개발자들과 소프트웨어 개발 조직들이 컴포넌트 기반 소프트웨어공학에 기대하고 있는 점은 이미 만들어진 유용한 컴포넌트들을 통합하는데 수반되는 노력이 전통적인 방법으로 소프트웨어를 개발하는데 드는 노력보다 적게 소요된다는 가정을 기초로 하고 있다[2]. 그러나 이는 결함이 없는 컴포넌트가 다른 컴포넌트들로 이루어진 시스템으로 통합되어질 때 예상한 기능을 수행하지 않을 수도 있다는 사실을 고려하지 않은 것이다[2]. 이것은 작성된 컴포넌트들이 여러 목적을 위해 작성되어졌을 수도 있으며, 여러 용법들을 가질 수

있기 때문이다. 현재 컴포넌트 기술은 상호 연결된 컴포넌트들의 구문적 적합성은 검증할 수 있으나 독립적으로 개발된 컴포넌트들이 통합되어질 때 애플리케이션의 기능이 정확히 동작하리라고 보증할 수는 없다[2]. 바꿔 말하면, 개개의 컴포넌트들이 통합되어 상호 연결되어진 경우 적합성은 점검하기 어렵다[10]. 따라서 컴포넌트 기반 소프트웨어의 상호운용성 테스트는 컴포넌트 기반 소프트웨어 개발의 성공을 위한 중요한 핵심 요소이다.

2.2 컴포넌트 기반 소프트웨어 테스팅

소프트웨어 공학자들은 컴포넌트 기반 시스템의 확인(validation)은 전통적으로 개발된 소프트웨어의 테스트와는 다르다는 사실을 인지하고 있다[6]. 컴포넌트 기반 소프트웨어의 테스트는 컴포넌트 기반 개발 과정에서 테스트 및 테스트와 관련된 모든 활동들을 수반한다. 즉, 컴포넌트를 개발하는 컴포넌트 제공자에게 수반되는 활동들과 제공받은 컴포넌트를 사용하여 애플리케이션을 구축하는 컴포넌트 사용자에게 수반되는 활동들을 모두 포함한다[6]. 컴포넌트 제공자와 컴포넌트 사용자는 같은 조직이 될 수도 있으며, 또한 다른 조직이 될 수도 있다.

제공자가 수행하는 테스트는 일반적으로 제공자 명세에 따라 컴포넌트의 내부 작업을 점검하는 것이다. 즉, 컴포넌트의 내부 로직, 데이터, 프로그램 구조 및 내부 알고리즘의 정확한 성능 등을 평가하는데 중점을 둔다. 컴포넌트 개발자 즉, 컴포넌트 제공자들은 컴포넌트의 내부 로직을 잘 알고 있기 때문에 제공자가 수행하는 테스트는 일반적으로 화이트박스 테스트 기법으로 수행된다[2]. 컴포넌트 제공자의 테스트 목적은 해당 컴포넌트의 기능 및 행위 명세에 부합하는 고품질의 재사용 가능한 프로덕트를 사용자에게 인도하는 것이다[2].

반면 컴포넌트 사용자가 수행하는 테스트는 일반적으로 애플리케이션을 구성하는 다른 컴포넌트들이 존재하는 프레임워크에 개개의 컴포넌트들이 적절한지를 평가한다. 이러한 테스트는 컴포넌트를 블랙박스 개체로 간주하고, 일반적으로 블랙박스 테스트 기법이 적용된다[2]. 컴포넌트 사용자의 목적은 컴포넌트 프레임워크에 기존 컴포넌트들을 통합시키는 것이며, 사용자 요구사항에 따라 여러 컴포넌트들이 정확히 협력하는지를 평가하는 것이다[2].

다음은 컴포넌트 기반 소프트웨어 테스트를 어렵게 만드는 중요 문제들을 요약한 목록이다[10,11].

- 새로운 환경에서의 컴포넌트 테스트

컴포넌트들은 제공자의 개발 환경에서 개발되며, 사용자에 의해 여러 배치 환경에서 재사용된다. 컴포넌트들은 종종 제공자가 전혀 예측하지 못한 환경에서 재사용되기 때문에 컴포넌트들 간의 상호운용성 검증은 컴포넌트 기반 소프트웨어 테스트에서의 중요한 사항으로 인식된다.

- 컴포넌트의 내부 구현에 대한 접근의 어려움

컴포넌트 기반 소프트웨어의 테스트에서 테스트 대상이 되는 개개 컴포넌트들의 낮은 가시성은 큰 문제이다. 이로 인해 컴포넌트에 대한 내부 정보에 접근 및 제어가 어려우며, 컴포넌트에 대한 테스트 또한 어렵다. 게다가 상업용 컴포넌트의 경우 대부분 컴포넌트의 인터페이스를 제외한 컴포넌트의 내부 정보는 얻어낼 수 없기 때문에 이러한 상황에서 컴포넌트들 간의 상호운용을 테스트하기 위한 체계적인 방법론에 대한 연구가 중요한 사항으로 인식되고 있다.

본 논문에서는 위의 사항들을 고려하여 컴포넌트 기반 소프트웨어 개발의 재사용성의 이점을 얻기 위한 전제조건인 컴포넌트 기반 소프트웨어의 상호운용성 테스트 기법을 제안하며, 컴포넌트 제공자 측면이 아닌 컴포넌트 사용자 측면에서 컴포넌트 기반 소프트웨어를 구성하는 컴포넌트들을 블랙박스 개체로 보고 블랙박스 테스트 기법을 이용한 상호운용성 테스트를 수행할 수 있는 기법을 제안한다.

3. 테스트 모델 생성

Griffeth는 Voip 시스템의 행위를 모델링하기 위하여 EFSM을 사용하였고 작성된 시스템 행위 모델을 기반으로 Voip 시스템의 상호운용성 테스트 기법을 제안하였다[12,13]. EFSM은 확장된 변수들을 갖는 유한 상태기계(Finate States Machines)이며, 유한 상태기계들의 최적화된 표현이다. 본 논문에서는 테스트 모델 생성 기법을 제안하기 위하여 먼저 EFSM 기반의 테스트 모델에 대하여 정의하고 테스트 모델 정의에 입각한 유스케이스 기반의 테스트 모델 생성 기법을 제안한다. 유스케이스 기반 접근 방법은 객체지향 패러다임에서 필요한 여러 정보를 포함시킬 수 있는 행위모델 다이어그램의 도출을 자연스럽고 용이하게 유도할 수 있고 유스케이스에는 상호운용 정보를 유추하기 위한 원시 정보가 포함되어 있기 때문에 본 연구에서는 자연어로 이루어진 유스케이스 명세를 기반으로 한 모델 생성기법을 제안한다.

그림 1은 테스트 모델 생성을 위한 프로세스를 보이고 있다.

3.1 테스트 모델 정의

모델이란 용어는 많은 컨텍스트(context)에서 사용되며, 여러 의미를 갖는다. Binder에 따르면 소프트웨어 테스팅에서 사용되는 모든 모델은 네 가지 주요 요인(주제, 관점, 표현, 기법)을 갖는다[14].

- 주제-생성되는 모델의 핵심아이디어로 일반적으로 SUT

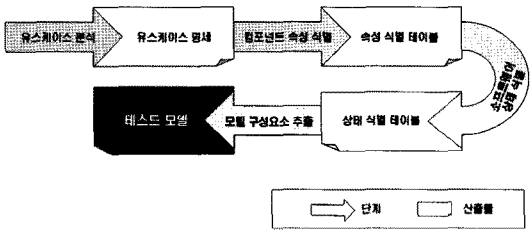

그림 1 테스트 모델 생성 프로세스

(System Under Test) 모델은 효율적인 테스트 케이스 선택에 도움을 준다.

- 관점-모델은 참조 프레임과 관련 이슈 및 정보의 식별을 도울 수 있는 원칙을 기반으로 해야 한다. 소프트웨어 테스팅 모델은 일반적으로 요구되는 행위를 표현하며, 버그가 의심되는 시스템의 측면들에 초점을 맞춘다. 소프트웨어 테스팅 관점에서의 모델은 테스트 케이스를 생성하고 평가하는데 필수적인 정보들을 포함해야 한다.
- 표현-모델링 기법은 특정 모델을 표현하는 방법을 포함해야 한다. 이것은 와이어프레임, CAD 모델 또는 기타 여러 형태가 될 수 있다.
- 기법-모델은 범용을 목적으로 정형 모델 또는 단순한 가공물 생성을 위하여 UML 표기법을 사용한 기법을 사용함으로써 개발될 수 있다.

본 논문에서는 이와 같은 모델의 주요 요인을 기반으로 하여 테스트 모델을 다음과 같이 정의한다.

정의 1. 테스트 모델은 테스트 케이스 생성에 필요한 시스템 정보를 유추할 수 있도록 구성된 모델로 모델 구성요소, 모델 시각화를 위한 다이어그램, 저장을 위한 형식을 갖는다.

테스트 케이스 생성은 본 절에서 정의하는 테스트 모델을 기반으로 생성된다. 본 논문에서 제안하는 테스트 모델을 정의 1에서 정의하고 있는 세 가지 요소 즉, 모델 구성요소, 다이어그램, 저장 형식 관점으로 기술하면 다음과 같다.

① 모델 구성요소

본 논문에서 제안하는 테스트 모델 M은 유한개의 상태들 간의 상태 전이를 기반으로 하고 있으며, 6개의 튜플로 구성된다. 다음은 EFSM 기반의 상태전이의 구성을 나타낸다.

$$M = (s_t,\ q_t,\ a_t,\ o_t,\ P_t,\ A_t, c_t)$$

s_t : 시작 상태

q_t : 종료 상태

a_t : 시스템에 가해지는 행위에 수반되는 외부 입력

o_t : 행위 결과로 발생하는 시스템의 응답

$P_t(\vec{x})$: 상태 전이 후 지켜져야 하는 조건을 나타내는 불린 타입의 컴포넌트 속성 변수들의 값

A_t : 컴포넌트 속성 변수 값에 의해 정의되는 동작

c_t : 상호운용 수반 여부를 표현하는 정보

이 중 c_t 항목은 상태 전이에 상호운용이 포함되어 있는지 여부를 표현하기 위한 요소로 Hao가 제안한 기법을 활용하였다. Hao는 상호운용이 포함된 전이는 'black'으로, 포함되지 않는 전이는 'white'로 구분하여 표현하였다[12]. 이러한 정보는 후에 테스트 케이스 생성에 있어서 상호운용성 테스트와 관련 없는 부분을 제거하기 위한 정보로 활용된다.

② 다이어그램

다이어그램은 테스트 모델의 구성요소들을 그래픽 요소들로 표현한 것으로 그림 2는 본 논문에서 제안하고 있는 테스트 모델에 대한 시각적 표현을 나타내는 상태 전이 다이어그램이다. 노드는 시스템의 특정 상태를 나타내며, 방향을 가지는 간선은 상태 전이를 나타낸다. 테스트 모델의 다이어그램이 가지는 의의는 테스트 케이스 생성을 위한 테스트 모델이 가지는 구성요소와 구성요소들 간의 관계 및 제약사항을 시각적으로 나타내 줌으로써 테스터가 테스트 모델을 이해하는데 도움을 준다는 것이다. 또한 테스트 시퀀스 자동 추출을 위한 자동화 도구의 자료구조로서의 의미를 갖는다.

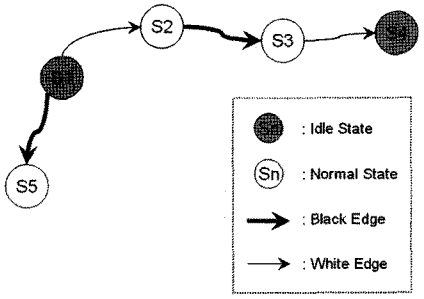

그림 2 모델 시각화를 위한 다이어그램

③ 저장 형식

소프트웨어 테스트 수행 시 신중히 고려해야할 문제점 중 하나는 테스트에 소모되는 인간의 노력도이다. 완전한 소프트웨어의 검증(verification) 및 확인(validation)은 소프트웨어 테스트의 주 목적이지만 소모되는 노력도가 크다면 그 의미를 잃는다. 따라서 소프트웨어 테스트 수행에 있어서 도구의 사용은 필수적이다. 도구를 이용하면 인간의 범할 수 있는 실수나 소모되는 노력도를 줄일 수 있다. 모델을 도구 내에서 활용하기 위해서는 모델을 도구 내에 저장하기 저장 형식이 필요하다. 본 논문에서는 최근 데이터 표현을 위해 많이 사용되는 XML을 이용하여 저장 형식을 정의한다. XML 형

```
<!-- ======= Start Entity Declaration ======= -->
<!-- ======= End Entity Declaration ======= -->
<!-- ======= Start Element Declaration ======= -->
<!ELEMENT EFSM (transition)+>
<!ELEMENT transition (from , trans_info , to)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT trans_info (input+ , output+ , predicate+ , action+ ,color )>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!-- ======= End Element Declaration ======= -->
<!-- ======= Start Attribute Declaration ======= -->
<!ATTLIST EFSM          e_id ID #REQUIRED>
<!ATTLIST transition id ID #REQUIRED>
<!-- ======= End Attribute Declaration ======= -->
```

그림 3 EFSM 상태전이 명세의 XML DTD

식의 정의를 선택한 이유는 XML로 정의된 데이터는 애플리케이션에서 활용하기 용이하기 때문이다. 그림 3은 본 논문에서 제안하는 테스트 모델을 저장하기 위한 XML DTD이다. 저장되는 정보는 컴포넌트 기반 소프트웨어의 상태와 상태 전이 관련 정보들로서 자동화 도구 구현 시 이와 같은 형식으로 저장된 XML 데이터를 입력으로 하여 그림 2와 같은 다이어그램 및 테스트 시퀀스를 자동 생성할 수 있다.

3.2 테스트 모델 생성기법

상호운용성 테스트는 통신분야와 웹 애플리케이션 분야에서 사용되어온 테스트 용어로 ETSI(European Telecommunications Standards Institute)는 '둘 이상의 통신하는 시스템 사이에 요구되는 기능들을 증명하기 위한 활동'으로 상호운용성 테스트를 정의하고 있다[15]. 본 논문에서는 이러한 정의를 기초로 하여 컴포넌트 기반 소프트웨어의 상호운용성 테스트를 다음과 같이 정의한다.

정의 2. 컴포넌트 기반 소프트웨어의 상호운용성 테스트는 컴포넌트들로 구성된 소프트웨어에서 둘 이상의 서로 협력하는 컴포넌트들 사이에 요구되는 기능들을 증명하기 위한 활동이다.

또한 본 논문의 기법 제안을 위하여 다음과 같이 가정한다.

가정 1. 소프트웨어를 구성하는 컴포넌트들은 블랙박스 개체이며, 최종 사용자의 행위를 통해서만 시스템에 접근할 수 있다.

가정 2. 컴포넌트 간 이벤트 흐름을 파악할 수 있는 컴포넌트 명세가 존재한다.

테스트 모델 생성은 그림 2에서 제시한 바와 같이 유스케이스 분석, 컴포넌트 속성 식별, 소프트웨어 상태 식별, 테스트 모델 생성 순으로 진행된다.

3.2.1 유스케이스 분석

일반적으로 유스케이스 명세는 수반되는 액터, 사전조건, 이벤트 흐름의 상세한 기술, 사후조건으로 구성된다. 상세한 기술은 이벤트의 주 흐름과 대안 흐름으로 나뉜다[14].

유스케이스 명세 중 테스트 모델 생성을 위하여 참조하는 부분은 주 흐름과 대안 흐름이다. 본 논문에서는 이후 산출물들과의 전후참조와 컴포넌트 간 이벤트 흐름을 보다 명확히 하기 위하여 다음과 같은 규칙을 통하여 주 흐름과 대안 흐름을 기술한다.

규칙 1. 주 흐름의 문장에는 접두어 M과 일련번호를 붙여 기술하고, 대안 흐름의 문장에는 접두어 A와 일련번호를 붙여 기술하며, 특정 이벤트 흐름에 부속된 이벤트 흐름은 하위 일련번호를 붙여 기술한다.

표 1은 Gao가 제안한 방법에 따라 인터넷 쇼핑몰의 '사용자 인증' 유스케이스 명세를 기술한 예이다. 본 논문에서는 표 1에 제시된 유스케이스를 토대로 제시한 테스트 모델생성 기법의 사례를 제시할 것이다.

3.2.2 컴포넌트 속성 식별

컴포넌트 속성 식별 단계에서는 유스케이스 명세의 주 흐름과 대안 흐름에 기술된 시나리오를 기반으로 소프트웨어를 구성하는 컴포넌트들의 속성들을 식별하여 속성 식별 테이블을 생성한다. 컴포넌트 속성은 다음과 같이 정의한다.

정의 3. 컴포넌트 속성은 특정 순간 해당 컴포넌트의 특성을 참 또는 거짓으로 표현 가능한 변수이다.

속성 식별 테이블의 각 항목에는 표 2와 같은 내용을 기입한다.

컴포넌트 속성을 식별하기 위하여 유스케이스 명세마다 하나의 속성 식별 테이블을 작성한다. 속성 식별 테이블 작성 규칙은 다음과 같다.

규칙 2. 유스케이스 명세의 'Main Flow'와 'Alternative Flow' 항목에 기술된 기 수행 단계 번호를

표 1 'User Authentication' 유스케이스 명세

Use case	U001. User Authentication
Brief description	사용자가 고객으로서 프로그램의 주요 기능을 사용할 수 있도록 사용자 인증을 처리한다.
Actors	고객
Preconditions	고객은 판매자의 웹페이지 첫 화면에 위치해 있어야 한다.
Main flow	M1. 고객은 'Log in' 페이지의 'name' 필드와 'password' 필드에 자신의 id와 패스워드를 입력한다. M2. 고객은 입력 후 로그인페이지의 'Submit' 버튼을 클릭한다. M2.1. 'Log in' 페이지는 'name' 필드에 입력된 내용의 고객이 존재하는지 'Customer' 컴포넌트에게 요청한다. M2.2. 'Customer' 컴포넌트는 'name' 필드에 입력된 내용과 일치하는 이름을 가진 고객들의 정보를 'Log in' 페이지에 보낸다. M2.3. 'Log in' 페이지는 얻어온 해당 고객들의 패스워드와 'password' 필드에 입력된 내용이 일치하는 고객이 존재하는지 확인한다. M2.4. 일치하는 고객이 존재하면 해당 고객의 쇼핑카트를 생성하도록 'Quote' 컴포넌트에게 요청한다. M2.5. 'Quote' 컴포넌트는 해당고객의 쇼핑카트를 생성한다. M2.6. 'Web Storefront' 페이지로 이동한다.
Alternative flow	A1. 고객이 'name' 필드와 'password' 필드를 모두 채우기 전에 'Submit' 버튼을 클릭 한다면 '회원가입' 폼으로 이동한다. A2. 'Log in' 페이지가 얻어온 해당 고객 정보의 패스워드와 'password' 필드에 입력된 내용이 일치하는 고객이 존재하지 않으면 에러메시지를 띄우고 'Log in' 페이지가 다시 화면에 표시된다.
Postconditions	유스케이스가 성공적이면 해당 고객의 쇼핑카트가 생성되고 'Web Storefront' 페이지가 화면에 표시된다.

표 2 속성 식별 테이블 구성 항목

항 목	설 명
Step No.	전후참조를 위해 기술되는 유스케이스 명세의 주 흐름 및 대안 흐름의 단계 번호
Corres. Comp.	속성 식별 대상이 되는 컴포넌트 식별자
Behavior	유스케이스 명세에 기술된 해당 단계에 대하여 발생한 행위
Alternative Condition	특정 조건에 따른 시스템 행위인 경우 해당 조건을 표현하기 위한 조건 항목
Attributes	유스케이스 명세에 기술된 주 흐름 및 대안 흐름 항목에 기술된 행위를 기준으로 행위가 일어난 전후를 고려하여 식별되는 해당 컴포넌트의 속성

'Step no' 항목에 기입하고, 단계 번호에 해당하는 행위를 'Behavior' 항목에 기입하며, 소프트웨어를 구성하는 전체 컴포넌트들 중 해당 이벤트 흐름에 수반되는 컴포넌트에 대하여 해당 컴포넌트의 ID를 부여하고 이를 'Corres. Comp.' 항목에 기입한다.

규칙 2는 소프트웨어를 구성하고 있는 컴포넌트들 중 유스케이스 명세의 'Main Flow'와 'Alternative Flow'에 기술된 각각의 행위와 관련된 컴포넌트들을 식별하는 규칙으로, 예를 들면 표 3에서 'Step no'가 M1인 행의 'Corres, Comp' 필드에 기술된 C1은 표 2의 'Main Flow'에 기술된 행위와 관련된 컴포넌트의 ID가 C1이라는 것을 의미한다.

규칙 3. 'Behavior' 항목에 기술된 행위 수행 전후 해당 컴포넌트의 상태 변화를 식별하여 컴포넌트의 특성을 참 또는 거짓으로 표현할 수 있도록 컴포넌트 속성 이름을 정의하여 '_component ID' 접미어를 붙여 'Attributes' 항목에 기술하고, 조건에 따라 발생 여부가 결정되는 행위가 존재하는 경우 해당 조건을 'Alternative Condition' 항목에 'A_' 접두어를 붙여 기술한다.

'Behavior'항목에 기술된 행위가 수행되고 나면 그 행위와 관련된 컴포넌트들은 행위 전후 상태가 다를 수 있다. 예를 들면 사용자 아이디와 패스워드를 입력하기 전후의 해당 컴포넌트는 다른 상태를 지닌다. 컴포넌트 속성은 이러한 행위 전후에 변화하는 컴포넌트 상태를 나타내기 위하여 부여한 이름으로 추후 컴포넌트들로 구성된 전체 소프트웨어의 상태를 식별하기 위하여 사용된다. 규칙 3은 이와 같은 소프트웨어를 구성하고 있는 컴포넌트들의 속성을 식별하기 위한 규칙을 나타낸다. 위에서 언급하고 있는 '컴포넌트 상태'는 해당 행위와 관련된 개개의 컴포넌트에 대한 상태를 의미하며 3.2.3절에서 언급하고 있는 컴포넌트들로 구성된 전체 소프트웨어의 상태를 의미하는 '소프트웨어의 상태'와는 다르다.

규칙 2~3을 기반으로 하는 속성 식별 테이블 작성 알고리즘은 그림 4와 같다.

표 3은 표 1의 유스케이스 명세에 대하여 위 속성 식별 테이블 작성 알고리즘에 의하여 작성한 속성 식별 테이블이다.

3.2.3 소프트웨어 상태 식별

소프트웨어 상태 식별단계에서는 식별된 컴포넌트들

속성 식별 테이블 작성 알고리즘

입력 : 유스케이스 명세
출력 : 속성 식별 테이블

1. 유스케이스 명세에 기술된 주 흐름 및 대안 흐름 에 대한 모든 단계들에 대하여 다음 작업을 반복한다.
   1.1 유스케이스 명세에서 분석하고자 하는 기능 수행 단계 번호를 'Step no.' 항목에 기입한다.
   1.2 유스케이스 명세 문서를 분석하여 행위에 참가하는 컴포넌트와 수행되는 행위를 각각 'Component' 항목과 'Behavior' 항목에 기입한다.
   1.3 'Behavior' 항목에 기술된 내용의 분석을 통해 다음과 같은 작업을 수행한다.
      1.3.1 특정 조건에 따른 시스템 행위인 경우 'Alternative Condition' 항목에 해당 조건을 표현할 수 있는 속성 이름을 기입한다(속성 이름 앞에 'A_' 형식의 접두어를 기술).
      1.3.2 기술된 시스템 행위의 발생 여부에 따른 해당 컴포넌트의 변화를 분석하여 속성 및 가능한 속성 값을 식별하고, 식별된 속성 및 속성 값을 'Attributes' 항목에 기술한다(행위의 주체가 되는 컴포넌트 아이디를 '_Compnent ID' 형식의 접미어를 붙여 기술).

그림 4 속성 식별 테이블 작성 알고리즘

표 3 'User Authentication' 유스케이스에 대한 속성 식별 테이블

Step No.	Corres. Comp.	Behaviors	Alternative Condition	Attributes
M1	C1	고객은 'Log in' 페이지의 'name' 필드와 'password' 필드에 자신의 id와 패스워드를 입력한다.	없음	dispLoginPageIdle_C1 filledLoginInfo_C1
M2	C1	고객은 입력 후 로그인페이지의 'Submit' 버튼을 클릭한다.	없음	clickedSubmit_C1
M2.1	C1 C2	'Log in' 페이지는 'name' 필드에 입력된 내용의 고객이 존재하는지 'Customer' 컴포넌트에게 요청한다.	없음	idle_C2 reqCustInfoToC2_C1
M2.2	C1 C2	'Customer' 컴포넌트는 'name' 필드에 입력된 내용과 일치하는 이름을 가진 고객들의 정보를 'Log in' 페이지에 보낸다.	없음	sendCustInfoToC1_C2
M2.3	C1	'Log in' 페이지는 얻어온 해당 고객들의 패스워드와 'password' 필드에 입력된 내용이 일치하는 고객이 존재하는지 확인한다.	없음	checkIdPsw_C1
M2.4	C1 C3	일치하는 고객이 존재하면 해당 고객의 쇼핑카트를 생성하도록 'Quote' 컴포넌트에게 요청한다.	A_authSuccess	reqCreateCartToC3_C1 idle_C3
M2.5	C3	'Quote' 컴포넌트는 해당고객의 쇼핑카트를 생성한다.	없음	createCart_C3
M2.6	C1	'Web Storefront' 페이지로 이동한다.	없음	dispWebstoreIdle_C1
A1	C1	고객이 'name' 필드와 'password' 필드를 모두 채우기 전에 'Submit' 버튼을 클릭한다면 '회원가입' 폼으로 이동한다.	없음	filledLoginInfo_C1 clickedSubmit_C1 dispRegForm_C1
A2		'Log in' 페이지가 얻어온 해당 고객 정보의 패스워드와 'password' 필드에 입력된 내용이 일치하는 고객이 존재하지 않으면 에러메시지를 띄우고 'Log in' 페이지가 다시 화면에 표시된다.	A_authSuccess	dispErrMsg_C1 dispLoginPageIdle_C1

의 속성들을 기반으로 하여 상태식별 테이블을 만들어 컴포넌트들로 구성되는 전체 애플리케이션의 상태들을 식별한다. 컴포넌트 기반 소프트웨어의 상태는 다음과 같이 정의한다.

정의 4. 컴포넌트 기반 소프트웨어의 상태는 몇몇 조건의 만족, 액션의 수행, 이벤트 실행 대기 등의 특정 순간에서의 조건을 의미하며, 이는 특정 순간에서 구성 컴포넌트들의 속성 값들의 조합으로 정의된다.

컴포넌트 속성 식별 테이블 하나당 하나의 상태 식별 테이블을 작성하며, 컴포넌트 속성 식별 테이블에 기술되어진 각 컴포넌트들의 속성들을 기반으로 가능한 속성 값들의 조합을 통하여 시스템의 상태를 식별한다. 상태의 식별은 컴포넌트 속성 식별 테이블의 'Step No.' 순으로 각 컴포넌트의 가능한 속성 값들의 조합을 찾아 상태들을 식별해 나간다. 상태 식별 테이블의 각 항목에는 표 4와 같은 내용을 기입하며, 상태 식별 테이블 작성규칙은 규칙 4, 5와 같다.

규칙 4. 컴포넌트 속성 식별 테이블의 각 컴포넌트 아이디를 'Component ID' 항목에 기입하고, 해당 컴포넌트에 해당하는 속성들을 'Attribute' 항목에 기입하며, 컴포넌트 속성 식별 테이블의 'Step No.' 순으로 행위를 분석하여 특정 순간 가능한 컴포넌트들 속성 값들을 'Attribute Value' 항목에 확장하여 기입해 나간다.

규칙 5. 'Alternative Condition'이 포함된 경우 해당 조건의 만족 여부에 따른 모든 경우를 고려하여 속성 값들의 조합을 생성하며, 가능한 각 컴포넌트 속성 값들의

표 4 상태 식별 테이블 구성 항목

항 목	설 명
Component ID	컴포넌트의 아이디를 기술
Attribute	해당 컴포넌트에 대해 컴포넌트 속성테이블에서 식별한 모든 속성들을 기술
Attribute Value	컴포넌트 속성 식별 테이블의 'Step No.' 순으로 분석을 통하여 각 컴포넌트의 가능한 속성 값들을 나열
State No.	'Attribute Value'의 각 열들(가능한 각 컴포넌트 속성 값들의 조합)을 하나의 시스템 상태로 식별하여 상태 번호 부여

상태 식별 테이블 작성 알고리즘

입력 : 속성 식별 테이블
출력 : 상태 식별 테이블

1. 컴포넌트 및 해당 컴포넌트들의 가능한 속성들을 'Comp. ID.' 항목과 'Attributes' 항목에 기입한다('Alternative Condition'에 해당하는 속성 값은 상태와 무관하기 때문에 해당 속성으로 고려하지 않음)
2. 컴포넌트 속성 식별 테이블의 'Step No.' 순으로 다음 단계를 반복한다.
   2.2 'Step No.' 순으로 'Behavior' 항목의 분석을 통하여 가능한 컴포넌트들의 모든 속성 값들의 조합을 생성하여 'Attribute values' 항목을 확장하여 기입해 나간다.
   2.3 'Alternative Condition'이 포함된 경우에는 해당 조건의 만족 여부에 따른 모든 경우를 고려하여 기입한다.
3. 'Attribute Values' 항목의 각 열(가능한 각 컴포넌트 속성 값들의 조합)을 하나의 시스템 상태로 식별하여 상태 번호를 부여한다.

그림 5 상태 식별 테이블 작성 알고리즘

조합을 하나의 상태로 식별하여 상태 번호를 부여한다.

규칙 4~5를 기반으로 하는 상태 식별 테이블 작성 알고리즘은 그림 5와 같다.

표 5는 표 3의 속성 식별 테이블과 표 1의 유스케이스 명세를 기반으로 그림 4의 상태 식별 테이블 작성 알고리즘을 이용하여 작성한 상태 식별 테이블이다. 표 3으로부터 상태식별 테이블을 얻는 과정은 먼저 표 3에서 식별한 컴포넌트들의 ID를 'Comp ID' 필드에 기술하고 해당 컴포넌트의 속성으로 식별된 속성들을 'Attribute' 항목에 기술한다. 이후 표 3에 기술한 'Step No.' 순으로 해당 행위가 수행되기 전후 각각의 컴포넌트들의 속성 값의 변화를 고려하여 'Attribute Values'에 값들을 기입해 나간다. 예를 들면 표 3의 Step No, 1에 기술된 행위가 발생하기 전 각각 컴포넌트의 속성 값들을 'Attribute Values' 항목의 1열에 기술하고 발생한 후 속성 값들을 2열에 기술한다. 이와 같은 방법으로 행위 수행에 따라 발생하는 컴포넌트 변화를 분석하여 다음 열들의 컴포넌트 속성 값들을 기입해 나가며, 행위 발생이 연속적인 경우는 연속적인 행위들을 통합하여 기입해 나간다. 이와 같이 작성한 상태 식별 테이블에서 소프트웨어의 상태는 특정 순간 소프트웨어를 구성하는 각각의 컴포넌트들이 갖는 속성 값들에 따라 정해지며, 표 5에서 'Attribute Values' 항목의 각 열이 소프트웨어의 상태가 되고 식별된 상태에는 상태 번호(State No.)를 부여한다.

표 5 'User Authentication' 유스케이스에 대한 상태 식별 테이블

Comp. ID.	Attributes	Attribute Values				
C1	dispLoginPageIdle_C1	T	F	F	F	F
	filledLoginInfo_C1	F	T	T	F	F
	clickedSubmit_C1	F	F	T	F	T
	reqCustInfoToC2_C1	F	F	T	F	T
	checkIdPsw_C1	F	F	T	F	T
	reqCreateCartToC3_C1	F	F	T	F	F
	dispWebstoreIdle_C1	F	F	F	T	F
	dispRegForm_C1	F	F	F	F	T
	dispErrMsg_C1	F	F	F	F	T
C2	idle_C2	T	T	F	T	T
	sendCustInfoToC1_C2	F	F	T	F	F
C3	idle_C3	T	T	F	T	T
	createCart_C3	F	F	T	F	F
State No.		S1	S2	S3	S4	S5

3.2.4 테스트 모델 생성

테스트 모델 생성 단계에서는 식별된 소프트웨어의 상태들을 기반으로 상태 전이를 추출하고 이를 통하여 3.1절에서 정의한 테스트모델의 구성요소와 저장형식, 다이어그램을 생성해 낸다.

앞서 기술한 유스케이스 명세, 속성 식별 테이블 및 상태 식별 테이블에서 추출된 정보를 이용하여 테스트 모델의 구성요소를 식별한다. 규칙 6~8은 테스트 모델 구성요소를 식별하기 위한 규칙이다.

규칙 6. 유스케이스 명세의 이벤트 흐름을 참조하여 상태 식별 테이블의 상태들에 대한 전이 정보를 생성하고, 상태들의 전이 순서에 따라 'From-State' 항목과 'To-State'을 기술한다.

규칙 7. 상태 전이에 수반되는 외부 입력을 'Input' 항목에 기술하고, 상태 전이 결과로 발생하는 시스템을 응답을 'Output' 항목에 기술하며, 전이 후 지켜져야 하는 조건을 상태식별 테이블의 컴포넌트들의 속성 값들을 이용하여 'Predicates' 항목에 기술한다.

규칙 8. 전이가 발생하기 위하여 수행되는 행위를 'Action' 항목에 기술하고, 상태 전이가 상호운용을 수반하면 'Color' 항목에 'white'를 기술하고 수반하지 않으면 'black'을 기술한다.

그림 6은 표 5의 상태 식별 테이블에 규칙 6~8을 적용하여 식별한 테스트 모델 구성요소를 그림 3에서 정의한 XML DTD 형식으로 나타낸 명세이다.

생성한 명세는 컴포넌트 기반 소프트웨어의 상태들과 상태 전이 정보들을 포함하고 있다. 따라서 포함하고 있는 정보를 토대로 상태 전이 다이어그램을 생성할 수 있다. 생성한 상태 전이 다이어그램은 테스트 시퀀스 생성을 위하여 사용된다.

본 논문에서 제안하는 테스트 모델의 시각적 뷰인 상태 전이 다이어그램은 테스트 시퀀스 생성에 있어서 상호운용과 관련이 없는 불필요한 테스트를 제외하기 위하여 상태를 'Idle' 상태와 'Normal' 상태, 그리고 전이를 'White' 간선과 'Black' 간선으로 구분하여 기술한다. 'Idle' 상태와 'Normal' 상태에 대한 정의는 다음과 같다.

정의 5. 'Idle' 상태는 시스템에 아무 행위도 일어나지 않고 있는 즉, 이벤트의 발생을 기다리고 있는 휴면상태이며, 'Normal' 상태는 시스템에 어떠한 조건이 변경되었거나 이벤트가 발생하여 시스템이 어떠한 행위를 수행하고 있는 상태이다.

'White' 간선과 'Black' 간선에 대한 정의는 다음과 같다.

정의 6. 'White' 간선은 컴포넌트들 사이에 상호운용이 발생하지 않은 상태 전이를 나타내며, 'Black' 간선은 컴포넌트들 사이에 상호운용이 발생한 상태 전이를 나타낸다.

상태 전이 다이어그램 생성을 위한 규칙은 다음과 같다.

규칙 9. 상태전이 명세의 'From-State' 항목과 'To-State' 항목에 기술된 내용에 따라 상태들과 상태들 간의 전이를 표기법에 따라 기술하고, 각 상태에 해당하는 컴포넌트들의 속성 값을 분석하여 'Idle' 상태와 'Normal' 상태를 해당 표기법에 따라 기술하며, 'Color' 항목의 값에 따라 간선을 'White' 간선과 'Black' 간선으로 구분하여 기술한다.

```
<transition id = "T1">
   <from>S1</from>
   <trans_info>
      <input>String name; String password</input>
      <output></output>
      <predicates>filledLoginInfo_C1=true</predicates>
      <actions>
         <act1>input 'name' field and 'password' field</act1>
      </actions>
      <color>White</color>
   </trans_info>
   <to>S2</to>
</transition>
<transition id = "T2">
   <from>S1</from>
   <trans_info>
      <input></input>
      <output>Display Error Message; Load Registration Form</output>
      <predicates>A_authSuccess=false; filledLoginInfo_C1=false;
                  clickedSubmit_C1=true; reqCustInfoToC2_C1=true;
                  checkIdPsw_C1=true; dispRegForm_C1=true;
                  dispErrMsg_C1=true</predicates>
      <actions>
         <act1>Click 'Submit'Button</act1>
      </actions>
      <color>Black</color>
   </trans_info>
   <to>S5</to>
</transition>
<transition id = "T3">
   <from>S2</from>
   <trans_info>
      <input>String name; String password</input>
      <output>Create Shopping Cart;Display 'Web Store'Page</output>
      <predicates>A_authSuccess=true; filledLoginInfo_C1=true;
                  clickedSubmit_C1=true; reqCustInfoToC2_C1=true;
                  checkIdPsw_C1=true; sendCustInfoToC1_C2=true;
                  createCart_C3</predicates>
      <actions>
         <act1>Click 'Submit' Button</act1>
      </actions>
      <color>Black</color>
   </trans_info>
   <to>S3</to>
</transition>
<transition id = "T4">
   <from>S3</from>
   <trans_info>
      <input></input>
      <output>Display 'Web Store' Page</output>
      <predicates>dispWebStoreIdel_C1=true</predicates>
      <actions>
         <act1></act1>
      </actions>
      <color>White</color>
   </trans_info>
   <to>S4</to>
</transition>
```

그림 6 XML 형식의 상태전이 명세

이와 같이 작성된 상태 전이 다이어그램과 EFSM 명세를 이용하여 테스트 시퀀스 및 테스트 케이스를 생성한다. 그림 7은 규칙 9를 적용하여 그림 6의 명세를 상태 전이 다이어그램으로 표현한 것이다.

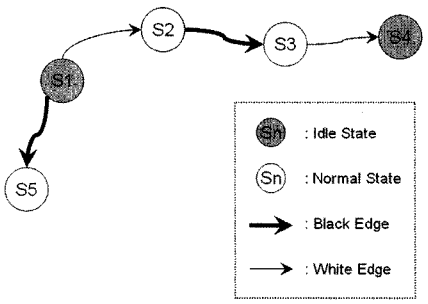

그림 7 그림 6에 대한 상태 전이 다이어그램

4. 테스트 케이스 생성 기법

본 논문에서 제안하는 테스트 케이스 생성의 목적은

컴포넌트간 상호운용성만을 점검하기 위하여 완전하고 최적화된 테스트 케이스들을 생성하는 것이다. 그림 8은 3장에서 제안한 기법에 따라 생성된 테스트 모델을 이용하여 상호운용성 테스트 케이스를 생성하는 프로세스를 나타낸다.

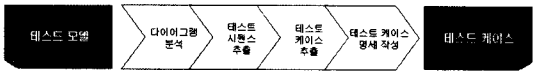

그림 8 테스트 케이스 생성 프로세스

4.1 상태 전이 다이어그램을 이용한 테스트 시퀀스 추출

테스트 시퀀스 생성을 위한 가장 중요한 작업은 상태 전이 다이어그램으로부터 비순환 경로를 찾아내는 것이다. 사이클이 존재하는 상태 전이 다이어그램에서 비순환 경로를 찾아내는 규칙은 다음과 같다. 그림 9는 SCC를 찾는 과정을 보이고 있다.

규칙 10. 상태 전이 다이어그램에서 반복되는 정점이 없는 가능한 모든 비순환 경로를 생성하고, 모든 쌍의 정점들 사이에 상호 방향 경로가 존재하는 최대 부분 그래프인 SCC(Strongly Connected Components)를 모두 찾는다.

규칙 11. 규칙 10에 의해 생성된 비순환 경로에 SCC를 합성하여 최종 경로 집합을 생성한다.

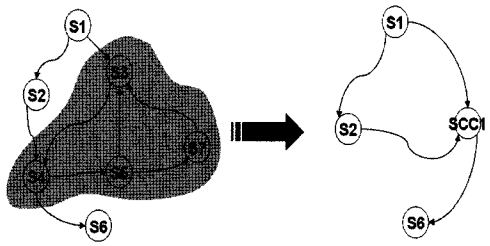

그림 9 도달성 그래프와 DAG

• 테스트 시퀀스 생성 알고리즘

Hao는 상태 전이 다이어그램으로부터 상호운용적인 요소만을 포함하는 테스트 시퀀스 생성을 위한 알고리즘을 제안하였다[12]. *본 논문에서 제안하고 있는 알고리즘은 상호운용과 관련 없는 휴면노드와 단말노드로의 화이트 전이를 제거하는 부분 고려하여 Hao의 알고리즘을 수정한 알고리즘이다.* 이 알고리즘을 기반으로 앞 절에서 기술한 상태 전이 다이어그램의 그래프 성질과 상호운용성 테스트를 위하여 EFSM에 포함된 정보를 이용하여 테스트 시퀀스를 생성하는 알고리즘을 제안한다. 그림 10은 상호운용성 검증을 위하여 상호운용이 존재하지 않는 부분을 다이어그램에서 제외하고, 규칙 2~규칙 3을 이용하여 테스트 시퀀스를 추출하기 위한 알고리즘을 기술하고 있다. 알고리즘에 기술된 슈퍼 간선은 다음과 같이 정의한다.

정의 7. 슈퍼 간선(super edge)은 루트 노드(root node)에서 최초 블랙간선(black edgy)이 나오는 노드(node)까지 직접 연결한 간선(edge)이다.

상태와 간선들을 제거하는 기준은 상태 전이의 컴포넌트간 상호운용 발생 여부와 상호운용을 포함하지 않는 전이라 하더라도 전후의 상호운용 관련 전이에 영향을 미치는지의 여부로 결정된다. 알고리즘에 기술된 간선 및 상태 제거 규칙은 다음과 같다.

규칙 12. 루트 노드(root node)로부터 최초 진출 블랙 간선(black edge)을 갖는 노드까지의 모든 화이트 간선(white edge)들은 슈퍼 간선(super edge)으로 대체하고, 휴면 노드(idle node)로의 모든 진입 화이트 간선(white edge)을 제거하며, 단말 노드(terminal node)로의 모든 진입 화이트 간선(white edge)을 제거한다.

상태 전이 다이어그램에서 화이트 간선은 상호운용이 없는 상태 전이를 나타낸다. 따라서 제거되어야 하지만 시작 상태 즉, 루트 노드에서 최초 상호운용이 있는 상태까지의 전이는 고려되어야 하므로 모든 화이트 전이들을 하나의 전이로 표현한다. 알고리즘의 1~4행은 이러한 과정을 나타낸다. 휴면 노드는 시스템에 어떠한 행위도 반영되지 않고 이벤트를 기다리고 있는 상태를 표현한 것으로 휴면 노드로의 화이트 전이는 컴포넌트간 상호운용과는 아무런 관련이 없기 때문에 해당 간선을 다이어그램에서 삭제한다. 알고리즘의 5행은 이러한 과정을 나타낸다. 위의 과정을 적용하고 나면 단말 노드들이 생성 되는데 단말 노드들로의 화이트 전이 또한 상호운용과는 관련이 없기 때문에 다이어그램에서 삭제한다. 알고리즘의 6행은 이러한 과정을 나타낸다. 알고리즘 7행은 규칙 4를 적용하여 생성한 상태 전이 다이어그램에 규칙 2~규칙 3을 적용하여 전체 비순환 경로를 생성하는 과정을 나타낸다. 이렇게 생성된 전체 비순환 경로가 테스트 시퀀스가 된다.

테스트 시퀀스 생성 알고리즘

상태 전이 다이어그램에서

1 진출 블랙 간선을 갖는 노드 v에서
2 루트 노드로부터 v로의 모든 화이트 경로에
3 루트 노드로부터 v로의 슈퍼 간선 추가
4 슈퍼 간선을 제외한 루트 노드로부터 v로의 모든 진출 화이트 간선 제거
5 휴면 노드로의 모든 진입 화이트 간선 제거
6 단말 노드로의 모든 진입 화이트 간선 제거
7 비순환 검색 알고리즘을 이용하여 전체 비순환 경로 생성

그림 10 테스트 시퀀스 생성 알고리즘

그림 11은 그림 9의 다이어그램에 그림 10 알고리즘을 적용하여 테스트 시퀀스를 생성하는 과정을 보인다.

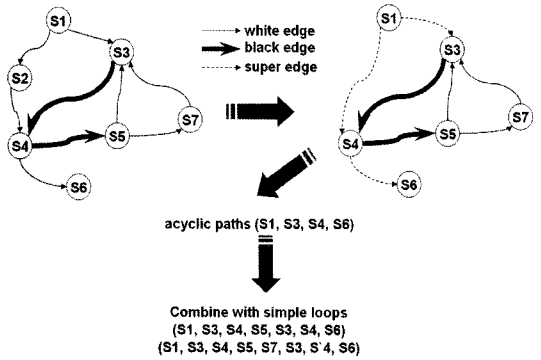

그림 11 테스트 시퀀스 생성 알고리즘을 이용한 테스트 시퀀스 생성

4.2 테스트 케이스 명세

테스트 케이스 명세는 일반적으로 표 6과 같은 정보를 포함한다[16]. 본 논문에서는 이와 같은 정보를 포함할 수 있도록 테스트 케이스 명세를 작성한다.

표 6 테스트케이스 명세에 포함되는 정보

테스트케이스 명세 식별자	테스트 케이스 명세 구분을 위하여 부여되는 식별자
테스트 항목	해당 테스트 케이스에 의해 수행되는 테스트 항목 및 특성
입력 명세	테스트 케이스 수행에 요구되는 입력
출력 명세	해당 테스트 항목에 요구되는 출력 및 특성
환경 요구사항	해당 테스트 항목에 대한 소프트웨어 및 하드웨어 제약사항
절차적 요구사항	해당 테스트 케이스 수행 절차상의 제약사항
테스트 케이스 간 종속성	해당 테스트 케이스에 선행되어 실행되어야 하는 테스트 케이스의 식별자 리스트

4.3 적용 사례

4.3.1 테스트 시퀀스 생성

그림 12는 제안한 기법의 적용사례를 보이기 위해 작성된 상태전이 다이어그램의 한 예이다.

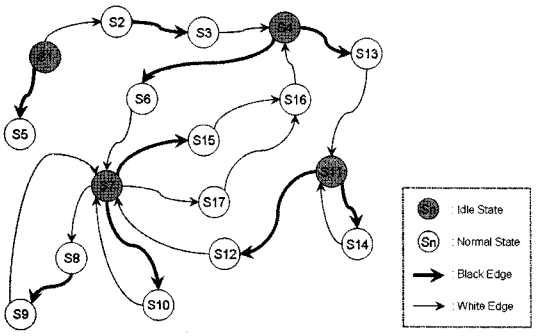

그림 12 상태 전이 다이어그램

그림 13은 테스트 시퀀스 생성을 위하여 테스트 시퀀스 생성 알고리즘의 1행에서 5행까지 수행한 결과를 나타낸다.

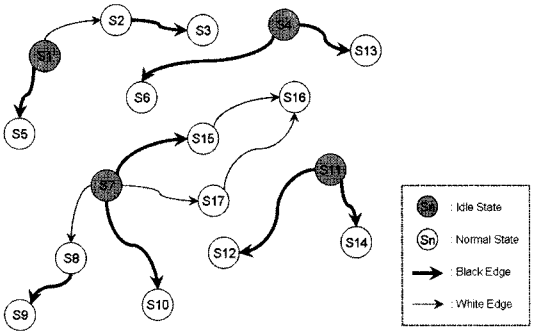

그림 13 휴면 노드로의 진입 화이트 간선 제거

그림 14는 테스트 시퀀스 생성 알고리즘의 6행을 실행한 결과이다.

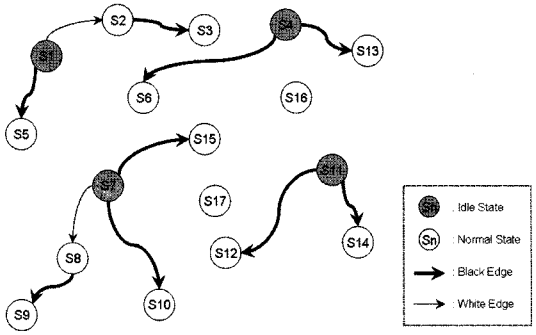

그림 14 단말 노드로의 모든 진입 화이트 간선 제거

테스트 시퀀스 생성 알고리즘의 7행에서 9행까지를 실행하여 테스트 시퀀스를 생성한 결과는 다음과 같다.

S1 → S2 → S3
S1 → S5
S4 → S6
S4 → S13
S7 → S8 → S9
S7 → S10
S7 → S15
S11 → S12
S11 → S14

4.3.2 테스트 케이스 명세 생성

표 7은 생성된 테스트 시퀀스를 기반으로 테스트 케이스를 생성하기 위하여 작성한 상태 전이 테이블의 일부이다.

표 7 상태 전이 테이블의 예

No	From State	Precondition	Input	Postcondition	To State
T01	S1	dispLoginPageIdel_C1=true	String name String password	filledLoginInfo_C1=true	S2
T02	S2	filledLoginInfo_C1=true	String name String password	A_authSuccess=true filledLoginInfo_C1=true clickedSubmit_C1=true reqCustInfoToC2_C1=true checkIdPsw_C1=true sendCustInfoToC1_C2=true createCart_C3	S3
…	…	…	…	…	…

표 8 테스트 케이스 명세

Test case identifier	T01	
State transition	S1 → S2	
Test item	Use case ID.	U001. User Authentication
	Interoperability	none
	Behavior	- 'Log in' 페이지의 'name'과 'password' 필드에 고객의 아이디와 패스워드를 입력함
Input	String name String password	
Output	없음	
Procedural requirements	- 고객은 판매자 웹페이지의 로그인 페이지에 위치해 있어야 함	
Intercase dependencies	없음	

테스트 케이스는 표 7의 상태 전이 테이블과 EFSM 명세를 기반으로 생성되며, 표 8은 'T01' 상태 전이에 의해 생성된 테스트 케이스 명세이다.

4.4 테스트 시퀀스 자동생성 도구

수작업이나 잘 정의된 테스트 제어 절차 없이 임의로 이루어지는 테스트가 갖는 비효율성을 극복하기 위하여 도구를 통한 자동화는 필수적이다. 본 절에서는 제안한 기법에 따라 구현한 테스트 모델 명세로부터 테스트 시퀀스를 생성해내는 자동화 도구 프로토타입을 소개한다.

본 논문에서 구현한 테스트 시퀀스 생성 도구는 그림 3에서 정의한 저장 형식에 따라 작성된 XML 형식의 테스트 모델 데이터를 입력받아 상태 전이 다이어그램을 구성하고 구성된 다이어그램과 제안한 알고리즘을 통해 테스트 시퀀스를 자동 생성하는 도구이다.

그림 15는 테스트모델 명세에 대한 XML 문서를 로드한 후 상태와 전이 정보를 추출하여 상태를 노드로 하고 전이를 간선으로 하는 상태 전이 그래프를 생성한 화면이다. 그림에서 머리가 빈 실선 화살표는 화이트 간선을, 머리가 채워진 실선 화살표는 블랙 간선을 의미한다.

그림 16은 최적화 규칙에 따라 불필요한 노드를 제거한 후 그래프를 재구성한 화면이다.

그림 17은 불필요한 간선 및 노드를 제거한 그래프를 기반으로 테스트 시퀀스를 생성한 결과 화면이다.

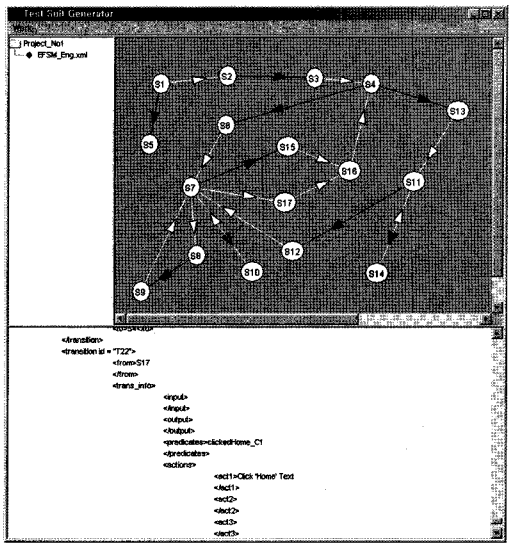

그림 15 테스트 모델 그래프 구성 화면

5. 연구의 의의

일반적으로 모델 기반 테스트 접근은 소프트웨어 산출물로부터 추출된 모델을 기반으로 자동화를 통하여 테스트 케이스를 쉽게 생성할 수 있도록 돕는다[8]. Arilo는 모델 기반 테스트과 관련하여 IEEEXplore, ACM

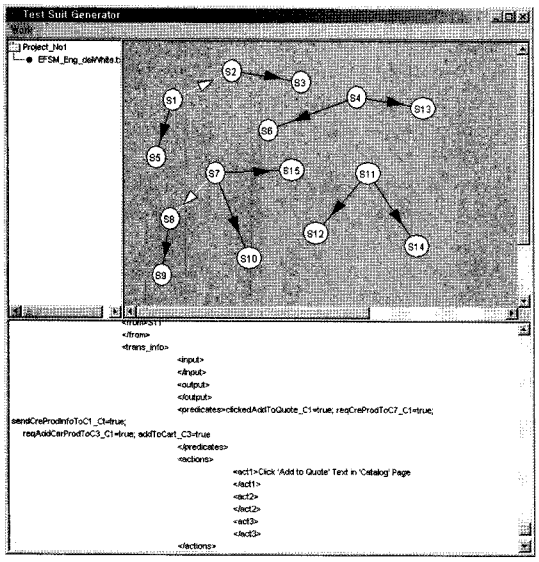

그림 16 불필요한 간선 및 노드 제거

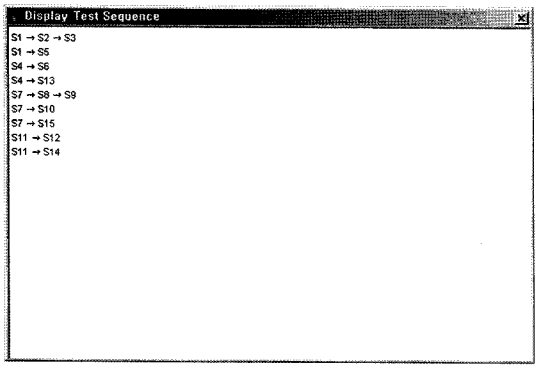

그림 17 테스트 시퀀스 생성 화면

Digital Library, Compendex EI, Inspec 그리고 Web of Science의 데이터베이스에서 인용이 많은 406개의 논문을 정제하여 UML 기반 접근, 비 UML 기반 접근 그리고 그 밖의 접근으로 분류하여 논문의 형태와 구축 패러다임 그리고 목표 도메인에 따른 카테고리를 작성하였다[18]. UML 기반 접근은 유스케이스, 상태다이어그램, 시퀀스 다이어그램, 협력다이어그램, 액티비티 다이어그램등 주로 UML의 하위 다이어그램을 이용하여 테스트 레벨과 도메인에 맞춰 예제 및 실험, 개념증명, 프로젝트 형태로 분류하였고[19-22], 비 UML 기반 접근은 주로 정형명세 및 유한 상태 기계(FSM)를 이용하여 예제 및 개념증명 형태로 분류하여 분석하였다[23-26]. 이러한 연구는 본 논문의 상호운용성 테스트라는 목적에 비춰 볼 때 보다 다양한 각도의 접근 방법을 제공한다.

상호운용성 테스트와 관련하여 Griffeth는 Voip 시스템의 행위를 모델링하기 위하여 EFSM(Extended States Machines)을 사용하였고 작성된 시스템 행위 모델을 기반으로 Voip 시스템의 상호운용성 테스트 기법을 제안하였다[12,13]. EFSM은 확장된 변수들을 갖는 유한 상태 기계(Finate States Machines)이며, 유한 상태기계들의 최적화된 표현이다. Griffeth는 EFSM을 이용하여 중복성이 제거된 테스트시퀀스 추출 알고리즘을 제안하였고 이를 기반으로 테스트 케이스를 최소화 하여 자동으로 생성할 수 있는 기법을 제안하였으며, 이에 따라 ITIS(Interoperability Testing Intelligent System) 도구를 구현하고 VoIP 시스템 상호운용성 테스트에 적용하여 그 실험 결과를 나타내었다[12,13]. Griffeth의 연구는 VoIP 시스템의 상호운용성 테스트 기법에 대한 기본 아이디어를 제공하지만 VoIP 시스템에 특화된 방법으로 도메인의 특성상 테스트 모델 생성을 위한 체계적 프로세스나 소프트웨어가 지니고 있는 다양한 요구사항들을 반영하기에는 적용상 어려움이 존재한다.

Brahim은 모델 기반 상호운용성 테스트를 위한 테스트 명세 방법을 제안하였으며 유스케이스 다이어그램, 시퀀스 다이어그램과 액티비티 다이어그램을 이용한 UML 개념과 U2TP(UML-2 Testing Profile)와 TTCN-3 (Testing and Test Control Notation 3rd version)을 사용하여 OffShore Framework을 제안하였으며 특히 초기 개발 단계에서 유용함을 제공한다[27]. 이러한 연구는 비즈니스 단계로부터 테스트까지 테스트 슈트명세 작성, 테스트 케이스 명세, 테스트 슈트 제어를 위해 기술적 요소의 매핑과 프로세스의 조작을 통해 모델 기반 테스트의 근거를 제시 한다. 그러나 자연어로부터 테스트 프로파일을 만들어 내는 과정에서의 프로세스가 명확히 기술되어있지 않다.

일반적인 테스트 케이스 모델은 UML 하위 다이어그램이나 테스트 프로파일을 이용하거나 상태 모델링이나 정형명세로부터 테스트 케이스를 추출한다. 이러한 방법의 이점은 모델이 갖는 자동화나 추상적 테스트 모델 접근이 용이하여 환영받는다. 유스케이스 기반 접근 방법은 객체지향 패러다임에서 필요한 여러 정보를 포함시킬 수 있는 행위모델 다이어그램의 도출을 자연스럽고 용이하게 유도할 수 있기 때문에 본 연구에서는 자연어로 이루어진 유스케이스 명세를 기반으로 하였으며, 이와 같이 자연어로부터 시작된 요구사항을 유스케이스 모델링까지 유도하기까지 수동적, 반자동적 방법을 이용한 연구가 여러 각도에서 이루어지고 있다.

본 논문에서는 유스케이스 명세를 기반으로 하여 상호운용성 테스트에 적합한 테스트 케이스 유도를 위한 모델을 생성하는 기법과 생성된 모델을 기반으로 테스트

케이스를 생성하는 기법을 제안하였으며, 기존 연구들에 비추어 본 논문의 연구 의의를 요약하면 다음과 같다.

- 컴포넌트의 낮은 가시성을 극복하기 위한 테스트 기법 제공
- 유스케이스 명세로부터 테스트 모델 생성까지의 체계적 프로세스 제공
- 컴포넌트 기반 소프트웨어의 상호운용성 테스트에 특화된 테스트 모델 및 중복 테스트 제거를 위한 테스트 시퀀스 생성 기법 제공

6. 결론 및 향후 연구

본 논문에서는 컴포넌트 기반 소프트웨어의 상호운용성 테스트를 위한 테스트 모델 생성 기법을 제안하고 생성된 모델을 기반으로 테스트 케이스 생성 기법을 제안하였다. 본 논문에서 제안한 테스트 케이스 생성기법의 의의는 다음과 같다.

첫째, 소프트웨어에 대한 완전한 정보를 얻기가 어려운 컴포넌트 기반 소프트웨어로부터 모든 종류의 입력을 제시하고, 관찰된 출력이 예상되는 출력과 일치하는지를 확인하는 블랙박스 형태의 테스트를 수행함으로써 컴포넌트의 낮은 가시성을 극복할 수 있다.

둘째, 테스트 케이스 생성 기법에 관한 연구들은 대부분이 완성된 테스트 모델을 기반으로 하고 있다. 본 논문에서 제안한 테스트 모델 생성기법은 소프트웨어 개발 시작단계에서 산출되는 유스케이스 명세로부터 테스트 케이스 생성의 기반이 되는 테스트 모델을 생성하는 체계적인 프로세스를 제공한다.

셋째, 상호운용성 테스트에 일반적인 테스트 모델을 사용할 경우 중복되거나 완전하지 못한 테스트 케이스를 생성하여 사용할 수 있다. 본 논문에서는 이와 같은 문제점을 해결하기 위하여 컴포넌트간 상호운용 정보를 포함할 수 있는 테스트 모델 생성 기법을 제안하였다.

본 논문에서 제시하고 있는 테스트 모델은 상호운용성 테스트 케이스 생성을 위한 입력으로 사용될 수 있다. 상호운용성 테스트 케이스 생성을 위하여 무엇보다 중요한 것은 필요한 시스템의 행위 정보를 정확히 포함하고 있는 테스트 모델을 추출하는 것이다. 그러나 테스트 모델 생성을 위한 초기 입력이 유스케이스 명세이므로 유스케이스 명세의 품질은 보다 완전한 형태의 모델 생성에 결정적인 역할을 한다고 볼 수 있다. 성공적인 상호운용성 테스트 케이스를 생성하기 위하여 가장 중요한 역할을 하는 것은 유스케이스 명세이다. 따라서 UML과 같은 반정형화된 모델링 기법이나 정형 명세언어를 이용하여 시스템을 분석한다는 것은 성공적인 테스트 케이스 생성에 매우 중요하다[17]. 따라서 이와 관련된 연구가 필요하다.

참 고 문 헌

[1] Peter Hersum, Oliver Sims, *Business Component Factory*, Wiley, 2000.

[2] Hans-Gerhard Gross, *Component-Based Software Testing with UML*, Springer, 2004.

[3] International Software Testing Qualification Board, *Standard glossary of terms used in Software Testing Version 1.2.*, 2006.

[4] J. Clark, C. Clarke, S. DePanfilis, G. Granatella, P. Predonzani, A. Sillitti, G. Succi, and T. Vernazza. "Selecting Components in Large COTS Repositories," Journal of Systems and Software, 2005.

[5] Object Management Group, *Model Driven Architecture-Resource Page*, Technical Report, http://www.omg.org, 2004.

[6] E.J. Weyuker. *The Trouble with Testing Components, In Component-Based Software Engineering, Heineman/Councill (Eds).* Addison-Wesley, 2001.

[7] Jos Warmer, Anneke Kleppe. *The Object Constraint Language Second Edition Getting your Models Ready for MDA.* Addison-Wesley, 2003.

[8] S.Dalal et al., "Model-Based Testing in Practice," Proc. 1999 Int'l Conf. Software Eng.(ICSE99), ACM Press, pp. 285-294, 1999.

[9] G. T. Heineman and W. T. Council (Eds). *Component-Based Software Engineering*, Addison-Wesley, Boston, 2001.

[10] J.Z. Gao, H.-S.J. Tsao, and Y. Wu. *Testing and Quality Assurance for Component-Based Software.* Artech house, 2003.

[11] J. Gao. "Challenges and Problems in Testing Software Components". In Workshop on Component-Based Software Engineering(ICSE 2000), Limerick, June 2000.

[12] R. Hao, D Lee, R. K. Sinha, N. Griffeth, "Integrated System Interoperability Testing with Applications to VoIP," IEEE/ACM Transactions on Networking, Vol.12, Issue 5, pp. 23-836, 2004.

[13] N. Griffeth, R. Hao, D. Lee, R. K. Sinha, "Interoperability Testing of VoIP Systems," Global Telecommunications Conference, Vol.3, pp. 1565-1570, 2000.

[14] Bind, *Testing Object-Oriented Systems: Models, Patterns and Tools.* Addison-Wesley, 2000.

[15] European Telecommunications Standards Institute (ETSI). *The Testing and Test Control Notation - Part 1.* Technical Report, 2003.

[16] IEEE. IEEE Standard for Software Test Documentation, IEEE Std 829. 2000.

[17] MIAO Hauikou and LIU Ling, "A Test Class Framework for Generating Test Cases from Z Specifications," 6th IEEE International Conference on Complex Computer Systems(ICECCS'00), Tokyo, Japan. pp. 164-171, 2000.

[18] Neto, A.D., Subramanyan, R., Vieira, M., Tra-

vassos, G.H., Shull, F., "Improving Evidence about Software Technologies: A Look at Model-Based Testing," Software, IEEE Volume 25, Issue 3, pp. 10-13, 2008.

[19] S. Ali et al., "A State-based Approach to Integration Testing Based on UML Models," Information and Software Technology, pp. 1087-1106, Nov. 2007.

[20] E. Bernard et al., "Model-Based Testing from UML Models," Proc. Int'l Workshop on Model-based Testing (MBT 2006), Lecture Notes in Informatics, vol. P-94, pp. 223-230, 2006.

[21] L.C. Briand, Y. Labiche, and J. Cui, Towards Automated Support for Deriving Test Data from UML Statecharts, tech. report, Carleton Univ., 2004.

[22] Y.G. Kim et al., "Test Cases Generation from UML State Diagrams," Software, vol. 146, no. 4, pp. 187-192, 1999.

[23] J. Chang and D.J. Richardson, "Structural Specification-based Testing: Automated Support and Experimental Evaluation," Sigsoft Software Eng. Notes, pp. 285-302. Nov. 1999.

[24] A. Paradkar, "Plannable Test Selection Criteria for FSMs Extracted From Operational Specifications," Proc. 15th Int'l Symp. Software Reliability Eng. (Issre 04), IEEE CS Press, pp. 173-184, 2004.

[25] A. Sinha and C. Smidts, "An Experimental Evaluation of a Higher-Ordered-Typed-Functional Specification-based Test-Generation Technique," Empirical Software Eng., Vol.11, No.2, pp. 173-202, 2006.

[26] L. Tan, O. Sokolsky, and I. Lee, "Specification-based Testing with Linear Temporal Logic," Proc. IEEE Int'l Conf. Information Reuse and Integration (IEEE IRI-2004), IEEE Press, pp. 483-498, Nov. 2004.

[27] Andaloussi, B.S., Braun, A., "A Test Specification Method for Software Interoperability Tests in Offshore Scenarios: A Case Study," Global Software Engineering, 2006. ICGSE '06. International Conference on, pp. 169-178, Oct. 2006.

유 철 중

1982년 전북대학교 전산통계학과 졸업(이학사). 1985년 전남대학교 대학원 계산통계학과 졸업(이학석사). 1994년 전북대학교 대학원 전산통계학과 졸업(이학박사). 1982년~1985년 전북대학교 전자계산소 조교. 1985년~1996년 전주기전여자대학 전자계산과 전임강사~부교수. 1997년~현재 전북대학교 공과대학 응용시스템공학부 정보공학전공 교수. 관심분야는 소프트웨어 개발 프로세스, 소프트웨어 품질, 컴포넌트 소프트웨어, 소프트웨어 메트릭스, 소프트웨어 테스팅, 소프트웨어 에이전트, 임베디드 소프트웨어, GNSS, GIS, 교육공학, 인지공학 등임

노 혜 민

2000년 전북대학교 컴퓨터과학과 졸업(이학사). 2002년 전북대학교 대학원 전산통계학과 졸업(이학석사). 2006년 전북대학교 대학원 컴퓨터통계정보학과 졸업(이학박사). 현재 전북대학교 BK21 전자정보 고급인력양성사업단 기금교수, 관심분야는 소프트웨어 개발 프로세스, 컴포넌트 기반 소프트웨어, 정형 명세 기법, 소프트웨어 테스팅 등임