상호운용성 테스트를 위한 테스트케이스 생성 기법

(Test Case Generation Technique for Interoperability Testing)

이 지 현 [†] 노 혜 민 [†] 유 철 중 [††] 장 옥 배 [††] 이 준 욱 [†††]
(Ji-Hyun Lee) (Hye-Min Noh) (Cheol-Jung Yoo) (Ok-Bae Chang) (Jun-Wook Lee)

요 약 네트워크 기술이 급격히 발전하면서 서로 다른 벤더들이 개발한 시스템들이 통합되거나 상호운용함으로서 특정 기능을 수행한다. 이 경우 통합된 시스템의 정확성을 보증하는 상호운용성 테스트는 필수적이다. 상호운용성 테스트는 다른 벤더들이 개발한 다른 시스템이 데이타를 공유하는 경우 소프트웨어나 하드웨어의 성능을 테스트하는 것이다. 많은 연구에서 시스템의 행위를 모델링하기 위하여 EFSM(Extended Finite State Machines)을 사용하고 있고, EFSM은 테스트케이스 생성 알고리즘의 입력으로 사용된다. 그러나 대부분의 연구들은 최적의 테스트케이스 생성 알고리즘에 대한 것들로서 이들 알고리즘의 입력이 되는 EFSM 명세를 생성하는 과정에 관한 연구는 찾아보기 힘들다. 본 논문은 상호운용성 테스트를 위한 테스트케이스를 생성하는 연구로서 요구사항 분석서로부터 EFSM 명세를 생성하는 방법을 제안하고, 테스트케이스 생성의 자동화를 위하여 제안한 기법으로 생성된 EFSM을 입력으로 하여 표준화된 테스트 케이스 및 슈트를 자동 생성하기 위한 테스트케이스 생성기의 프로토타입을 구현한다. 또한 프로토타입 구현에 적용된 이론적 배경 및 알고리즘을 상세히 설명한다.

키워드 : 상호운용성 테스트, 테스트케이스, 소프트웨어 공학

Abstract With the rapid growth of network technology, two or more products from different vendors are integrated and interact with each other to perform a certain function in the latest systems. Thus, interoperability testing is considered as an essential aspect of correctness of integrated systems. Interoperability testing is to test the ability of software and hardware on different machines from different vendors to share data. Most of the researches model communication system behavior using EFSM(Extended Finite State Machines) and use EFSM as an input of test scenario generation algorithm. Actually, there are many studies on systematic and optimal test case generation algorithms using EFSM. But in these researches, the study for generating EFSM model which is a foundation of test scenario generation isn't sufficient. This study proposes an EFSM generating technique from informal requirement analysis document for more complete interoperability testing. and implements prototype of Test Case Generation Tool generating test cases semi-automatically. Also we describe theoretical base and algorithms applied to prototype implementation.

Key words : Interoperability Testing, Test Case, Software Engineering

1. 서 론

현대 시스템들은 각종 네트워크 및 시스템간의 연동이 필수적인 시스템으로서 체계적이고 효율적인 상호운용성 테스트가 요구되는 시스템이다. 상호운용성 테스트는 다른 벤더들이 개발한 다른 시스템이 데이타를 공유하는 경우 소프트웨어나 하드웨어의 성능을 테스트하는 것이다. 상호운용성 테스트를 위한 테스트케이스 선정에 있어 수작업이나 잘 정의된 테스트 제어 절차 없이 임의로 이루어지는 테스트는 중복되거나 완전하지 못한 테스트케이스로 인한 비용낭비와 내재된 오류를 검출해내지 못하는 상황이 발생할 수 있다.

이러한 상호운용성 테스트의 부정확성 및 비효율성을 극복하기 위하여 테스트케이스 생성을 위한 체계적

† 학생회원 : 전북대학교 컴퓨터과학과
puduli@chonbuk.ac.kr
hmino@chonbuk.ac.kr
†† 종신회원 : 전북대학교 컴퓨터과학과 교수
cjyoo@chonbuk.ac.kr
okjang@chonbuk.ac.kr
††† 정 회 원 : 한국전자통신연구원 텔레매틱스연구단 연구원
junux@etre.re.kr
논문접수 : 2005년 5월 30일
심사완료 : 2005년 11월 16일

인 기법에 대한 정의와 테스트케이스 생성 과정을 일부 자동화하는 도구들에 대한 연구가 진행되어왔다[1,2].

그리고 많은 연구에서 시스템의 행위를 모델링하기 위하여 EFSM을 사용하고 있고 EFSM은 테스트케이스 생성 알고리즘의 입력으로 사용된다[3-5]. 따라서 EFSM 명세의 정확성 여부가 생성된 테스트의 정확성에 직접적인 영향을 미친다고 볼 수 있다. 그러나 대부분의 연구들이 최적의 테스트케이스 생성 알고리즘에 대한 것들로서 이들 알고리즘의 입력이 되는 EFSM 명세를 생성하는 과정은 언급되어 있지 않다.

본 논문은 상호운용성 테스트를 위한 테스트케이스를 생성하는 연구로서 요구사항 분석서로부터 EFSM 명세를 생성하는 방법을 제안한다. 그리고 테스트케이스 생성의 자동화를 위하여 제안한 기법으로 생성된 EFSM을 입력으로 하여 불완전한 테스트케이스로 인한 오류 검출의 실패를 최소화하고 중복된 테스트케이스로 인한 비용 낭비를 최소화할 수 있는 표준화된 테스트 케이스 및 슈트를 자동 생성하기 위한 테스트케이스 생성기의 프로토타입을 구현한다. 또한 프로토타입 구현에 적용된 이론적 배경 및 알고리즘을 상세히 설명한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 테스트케이스 생성 알고리즘의 입력으로 사용되는 EFSM과 테스트케이스 자동생성 도구 구현에 적용된 이론 및 알고리즘에 대하여 기술한다. 3장에서는 요구사항 분석서로부터 EFSM을 생성하는 방법을 제안하고, 제안한 EFSM을 입력으로 하여 테스트케이스를 자동 생성하는 프로토타입 구현에 관하여 기술한다. 4장에서는 3장에서 제안한 EFSM 작성 기법 및 테스트케이스 자동 생성 기법을 통하여 실제 요구사항 분석서로부터 테스트케이스를 생성하는 사례를 제시한다. 마지막으로, 5장에서는 연구결과에 대한 토의를 기술하고 6장에서 결론과 향후 연구과제를 제시한다.

2. 적용 이론 및 알고리즘

2.1 EFSM

시스템은 유한개의 상태를 가지며 입력에 따라 다른 상태로 전이하고 상태전이의 결과로서 출력을 생성한다. 이러한 시스템의 상태전이를 모델링하기 위하여 유한상태기계(Finite State Machines)가 사용되어왔다[6]. 그러나 FSM은 입력과 출력에 의해서만 상태전이를 표현하므로 변수들과 오퍼레이션을 포함하고 있는 시스템을 명세하기에는 충분하지 못하다. 따라서 FSM에 행위(action)를 추가하여 상태전이를 기술하는 EFSM (Extended Finite State Machines)이 사용되고 있다.

EFSM은 벡터로 정의되는 유한개의 변수들의 집합($\vec{x}$ = (x_1, ..., x_k))을 FSM에 추가한 정형 모델이다. EFSM은 입력심볼, 출력심볼, 상태, 변수, 전이의 5개의 튜플들로 이루어지며 다음과 같은 형태의 모델이다.

M = (I, O, S, $\vec{x}$, T)
I: Input symbols
O: Output symbols
S: States
$\vec{x}$: Variables(x_1, ..., x_k)
T: Transitions

EFSM은 한 상태에서 다른 상태로 전이하면서 변수들의 값을 변경하며 전이 집합 T내의 각 전이 t는 시작상태, 종료상태, 입력, 출력, 술어, 행위의 6개의 튜플로 이루어지며 그 구성은 다음과 같다.

t = (s_t, q_t, a_t, o_t, P_t, A_t)
s_t: start state
q_t: end state
a_t: input
o_t: output
P_t: $P_t(\vec{x})$는 변수(속성) $\vec{x}$값으로 TRUE, FALSE를 판별하는 술어
A_t: $A_t(\vec{x})$는 변수(속성) $\vec{x}$값에 의해 정의되는 행위

본 논문에서 EFSM 모델은 상호운용하는 구성요소들을 명세하기 위하여 사용되며, XML 형식으로 기술한다. XML은 어떤 플랫폼에서나 가독성 있는 포맷을 제공하기 때문에 테스트케이스 생성 도구의 입력으로 활용하기 용이하기 때문이다. 그리고 EFSM 모델은 상태전이그래프로 표현된다.

2.2 테스트 시퀀스 생성 기법

본 절에서는 [7,8]에서 VoIP 시스템의 상호운용성 테스트케이스를 생성하기 위하여 사용한 Adequate-Coverage 알고리즘을 설명한 후 테스트 시퀀스들을 생성하는 일련의 과정을 기술한다. 먼저 EFSM 모델을 통하여 도출한 상태전이를 기술하는 도달성 그래프 G로부터 전체 비순환 경로를 검색하는 과정을 기술한 후 Adequate-Coverage 알고리즘을 적용하여 테스트 시퀀스를 생성하는 과정을 설명한다.

2.2.1 비순환 경로 생성

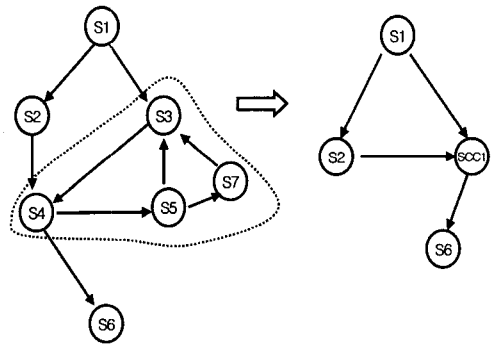
상태전이그래프 G로부터 비순환 경로를 검색하는 과정은 표 1과 같이 '비순환 경로 생성', '단순 사이클 생성', '비순환 경로와 단순 사이클의 결합'의 3단계로 나누어 볼 수 있다.

단계 1에서는 먼저 상태전이그래프로부터 SCC(strongly connected components : 강한 연결 요소)를 검색하는 일을 수행한다. SCC는 사이클을 포함하므로 SCC를 통

표 1 비순환 경로 검색 단계[2]

단계	설 명
1	반복되는 정점이 없는 가능한 모든 비순환 경로 생성
2	소규모의 다른 사이클을 포함하지 않는 가능한 모든 단순 사이클 생성
3	단계 1의 경로들과 사이클들을 최종 경로 집합과 결합

하여 단순 사이클 생성할 수 있을 뿐만 아니라 상태전이그래프로부터 사이클을 포함하고 있는 않은 그래프인 DAG(directed acyclic graph)를 얻을때 용이하다는 장점이 있다. 그런 다음 SCC를 노드로 하고, 본래의 상태전이그래프에 있는 노드 u에서 노드 v로의 간선이 u가 속해있는 SCC에서 v가 속해있는 SCC로의 간선이 되는 DAG를 작성한다. 검색한 SCC는 단순 사이클을 찾는데 사용되고 SCC를 포함하고 있는 DAG는 비순환 경로를 찾는데 사용된다. 그림 1은 도달성 그래프로부터 SCC를 검색한 후 DAG를 생성한 결과를 보여주고 있다.



(a) 도달성 그래프 (b) (a)에 대한 DAG

그림 1 도달성 그래프와 DAG

그림 1의 (b)와 같은 DAG로부터 전체 비순환 경로를 검색하기 위하여 PATHS-IN-DAG 알고리즘을 적용한다. PATHS-IN-DAG 알고리즘은 다음과 같다.

PATHS-IN-DAG 알고리즘

하나의 루트 노드(source node)와 여러 개의 단 노드(sink node) 가 있는 DAG G'에서

1 G에 있는 노드들을 위상 정렬
2 for i=n-1, ..., 0
3 v_i가 단 노드(sink node)이면
4 $p(v_i) = \{\Lambda\}$;/* 경로가 없는 단일 집합*/
5 아니면
6 v_i에서 나가는 간선이 w_1, ..., w_r이라고 가정하면
7 $p(v_i) = \bigcup_{j=1}^{r} (v_i, w_j)\ p(w_j)$;
8 return $p(v_0)$

그림 1의 예에서 DAG (b)를 위상 정렬하면 n=4이며 G' = {S1, S2, SCC1, S6}이 된다. G'에 PATHS-IN-DAG 알고리즘의 2행에서 7행을 수행하면 전체 비순환 경로 집합 P' = {(S1, S2, SCC1, S6), (S1, SCC1, S6)}을 얻는다.

다음으로 단계 2에서 설명하고 있는 단순 사이클을 찾기 위한 자료 구조인 Next-Transition-Tree를 작성한다. 임의의 노드 v에 대한 Next-Transition-Tree $T(v)$는 v로부터 SCC에 있는 다른 노드로의 모든 가능한 비순환 경로를 저장하며 트리의 루트 노드는 v가 된다. 일반적으로 임의의 노드 u의 자식 노드는 노드 u로부터 방향성 간선을 갖는 SCC 내의 모든 노드가 된다. 그러나 자식 노드가 루트 노드 v로부터 u로의 경로에 나타났다면 해당 노드는 트리에 포함되지 않는다. 한 노드는 트리에 여러 번 나타날 수 있지만 트리 경로에서는 그렇지 않다.

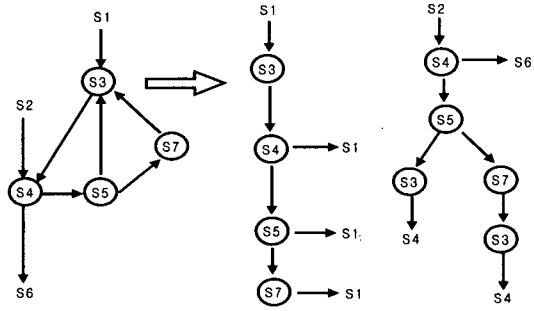


그림 2 그림 1에 대한 Next-Transition-Tree[2]

Next-Transition-Tree는 그래프에 있는 모든 노드에 대하여 작성하는 것이 아니라 SCC와 관련된 일부 노드에 대해서만 작성한다. 그림 2는 그림 1에 있는 상태전이그래프에 대한 Next-Transition-Graph로서 SCC상에 존재하는 노드 S3과 S4에 대한 Next-Transition-Tree이다.

전체 비순환 경로 검색 알고리즘

상태전이그래프 G에서

1 그래프 G에서 전체 SCC 검색
2 각 SCC에서
3 해당 SCC의 각 노드에서
4 Next-Transition-Tree 생성
5 DAG인 G'를 생성하기 위하여 G의 각 SCC를 축소
6 전체 비순환 루트 노드 - 단 노드 경로들인 P'를 생성하기 위하여 G'에 PATHS-IN-DAG를 적용
7 P'의 각 경로 p'에서
8 비순환 경로 집합인 P를 생성하기 위하여 p'에 있는 노드와 간선들을 G에 있는 경로들과 노드로 대체

그림 2에서 노드 S3이나 S4를 포함하는 단순 사이클 추출하면 (S3, S4, S5)과 (S3, S4, S5, S7)(또는 (S4, S5, S3), (S4, S5, S6, S3))가 된다.

다음은 DAG로부터 비순환 경로를 검색하기 위한 '전체 비순환 경로 검색' 알고리즘이다.

그림 1의 DAG에 있는 모든 비순환 경로를 찾은 후 단계 3에서 설명한 바와 같이 본래의 상태전이그래프 G의 노드와 경로로 대체하면 그림 3의 전체 비순환 경로와 같은 전체 비순환 경로를 얻을 수 있다.

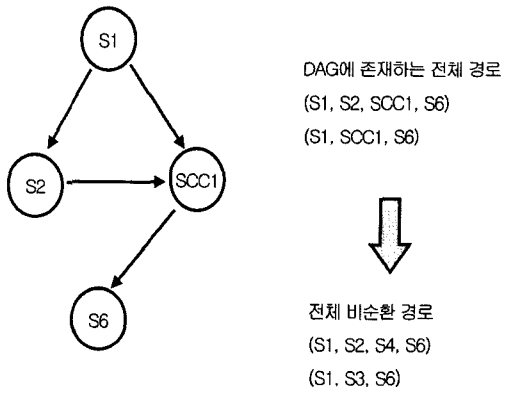


그림 3 전체 비순환 경로 생성

'전체 비순환 경로 검색' 알고리즘의 1행에서 6행을 수행하면 P' = {(S1, S2, SCC1, S6), (S1, SCC1, S6)}이므로 7행과 8행에 따라 SCC를 본래의 상태전이그래프 G에 있는 노드로 대체하면 P = {(S1, S2, S4, S6), (S1, S3, S4, S6)}이 된다.

2.2.2 테스트 시퀀스 생성

본 장에서는 Adequate-Coverage 알고리즘에 따라 실제 테스트 시퀀스를 생성하는 방법을 기술한다. 먼저 Adequate-Coverage 알고리즘은 다음과 같다.

Adequate-Coverage 알고리즘

상태전이그래프 G에서

1 진출 블랙간선을 갖는 노드 v에서
2 루트 노드로부터 v로의 모든 화이트 경로에
3 루트 노드로부터 v로의 슈퍼간선 추가
4 슈퍼간선을 제외한 루트 노드로부터 v로의 모든 진출 화이트간선 제거
5 진입 블랙간선을 갖는 각 노드 v에서
6 v로부터 단 노드로의 모든 화이트경로에
7 v로부터 해당 단 노드로의 슈퍼간선 추가
8 슈퍼간선을 제외한 단 노드로의 모든 진입 화이트간선 제거
9 비순환 검색 알고리즘을 이용하여 전체 비순환 경로 생성(단, 슈퍼간선은 화이트간선이나 다른 슈퍼간선 바로 앞이나 뒤에 오지 않는다)
10 각 슈퍼간선을 최단 본래의 그래프 G에 있는 화이트경로로 대체
/* 루프가 있을 수 있음 */
11 루프 제거

그림 4는 그림 1의 상태전이그래프 G에서 Adequate-Coverage 알고리즘의 1행에서 8행에 따라 비순환 경로를 생성하고 이 경로에 단순 사이클을 합성하여 테스트 시퀀스를 생성하는 과정을 보여주는 그림이다.

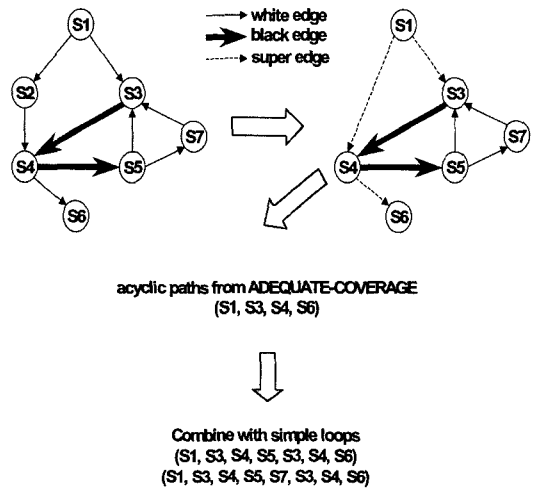


그림 4 Adequate-Coverage 알고리즘을 적용한 테스트 시퀀스 생성

그림 3에서 설명한 전체 비순환 경로 P에서 한 경로 (S1, S3, S6)가 제외된 이유는 S1과 S3 사이의 간선과 S3과 S6 사이의 간선이 모두 슈퍼간선이기 때문이다 (Adequate 알고리즘의 9행에 기술된 바와 같이 슈퍼간선은 화이트간선이나 다른 슈퍼간선 바로 앞이나 뒤에 오지 않아야 하므로).

상태전이그래프 G의 전체 비순환 경로 집합 P = {(S1, S3, S4, S6)}이므로 그림 2의 Next-Transition-Graph를 통하여 검색한 단순 사이클과 결합하면 테스트 시퀀스 집합 T = {(S1, S3, S4, S5, S4, S6), (S1, S3, S4, S5, S7, S3, S4, S6)}이 된다.

3. 테스트케이스 생성 기법

3장에서는 상호운용성 테스트를 위한 테스트케이스 생성 기법에 대하여 기술한다. 3.1절에서는 요구사항 분석서로부터 EFSM 명세를 생성하는 EFSM 기반 행위 모델링 기법을 제안하고, 3.2절에서는 테스트케이스 생

성의 자동화를 위하여 EFSM을 입력으로 하여 표준화된 테스트 케이스 및 슈트를 자동 생성하기 위한 테스트케이스 생성기의 프로토타입 및 프로토타입 구현에 적용된 이론적 배경 및 알고리즘을 상세히 설명한다.

그림 5는 테스트케이스를 생성하여 저장하는 전체 워크플로우를 나타내고 있다.

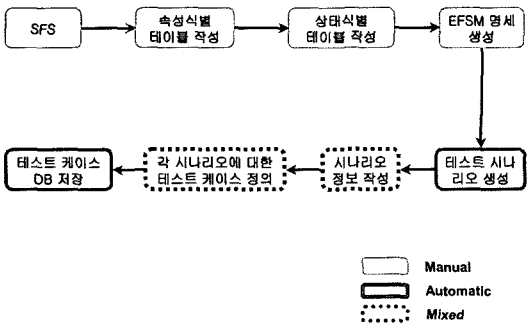


그림 5 테스트케이스 생성 워크플로우

3.1 EFSM 기반 행위모델링 기법

본 연구에서는 시스템의 행위를 EFSM을 기반으로하여 모델링하기 위하여 상호작용하는 구성요소들로 이루어진 시스템의 유스케이스 다이어그램, 유스케이스 문서 등을 통해 시스템 구성요소들의 구성 및 주요 기능을 분석하고자 한다. 그리고 이를 바탕으로 시스템 구성요소들 간의 시퀀스 다이어그램을 작성함으로써 시스템의 행위에 대한 정보를 추출하고자 한다. 그러나 요구사항 명세서가 자연어 또는 불명확한 다이어그램 형식으로 기술되어 있어 추상적이거나 모호성을 내포하고 있어서 행위 정보를 좀더 구체적이고 명확하게 표현하기 위해 SFS(Structured Functional Specification: 구조화된 기능 명세)라는 문서 형식을 정의하여 작성한 후 입력으로 사용한다.

그리고 SFS를 통하여 각 구성요소의 행위를 결정하는 속성(Attribute, Variable)을 식별하고 가능한 속성 값들을 기술하기 위해 컴포넌트 속성식별테이블이라는 문서 형식을 정의하여 작성한다. 작성된 속성식별테이블로부터 후보 속성들 및 관련된 컴포넌트, 가능한 속성의 값들을 식별함으로써 시스템의 가능한 상태들을 추출하는 상태식별테이블이라는 문서 형식을 정의하고 작성한다.

3.1.1 SFS 작성

자연어 또는 다이어그램 형식으로 기술되어 있는 추상적이거나 모호성을 내포하고 있는 행위 정보를 좀더 구체적이고 명확하게 표현하기 위해 SFS(Structured Functional Specification: 구조화된 기능 명세)라는 문서 형식을 정의하여 작성한다. SFS는 표 2와 같은 항목들로 구성된다.

3.1.2 속성식별테이블 작성

SFS 문서 하나당 하나의 속성식별테이블을 작성한다. SFS 문서의 각 기능 수행 단계들을 분석하여 시스템 구성요소의 속성을 식별하고 가능한 속성 값들을 기반으로 컴포넌트의 상태 이름을 정의한다. 이러한 일련의 과정을 SFS 문서에서 하나의 기능 수행 단계를 단위로 하여 식별해 나간다. 속성식별테이블의 각 항목에는 표 3과 같은 내용을 기입한다.

속성식별테이블은 다음과 같은 순서에 따라 작성한다.

표 2 SFS 구성 항목

항 목	설 명
Step	시퀀스 다이어그램에서의 단계를 기술한다.
Component ID	기능 단계의 주체가 되는 구성요소 ID를 기술하고 괄호 안에 관련 구성요소 ID를 병행 기술한다.
Precondition	해당 단계가 진행되기 전에 만족되어야 하는 시스템의 조건을 기술한다.
Input	해당 단계가 진행되기 위한 외부 입력을 기술한다.
Behavior	사전조건이 만족되고 해당 입력이 진행된 경우 시스템의 행위를 기술한다.
Postcondition	해당 단계가 진행된 후 보증해야 하는 시스템의 조건을 기술한다.
Output	각 단계의 가능한 출력값을 기술한다.

표 3 속성식별테이블 구성 항목

항 목	설 명
Step No.	전후참조를 위해 기술되는 SFS의 Step No.
Component	속성 식별 대상이 되는 구성요소 이름
Behavior	SFS에 기술된 해당 단계에 대한 시스템의 행위
Attribute	SFS의 Behavior 항목에 기술된 시스템 행위를 기준으로 행위가 일어난 전후를 고려하여 해당 구성요소의 속성 식별
Component State	SFS의 Behavior 항목에 기술된 시스템 행위를 기준으로 행위가 일어난 전후의 속성값 변화에 따른 컴포넌트 상태의 정의

속성식별테이블 작성 규칙

Input : SFS
Output : Attribute Identification Table

1. SFS에 기술된 해당 기능에 대한 모든 단계들에 대하여 다음 작업을 반복한다.
   1.1 SFS 문서에서 분석하고자 하는 기능 수행 단계 번호를 Step no. 항목에 기입한다.
   1.2 SFS의 구성요소 항목과 행위 항목을 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 Thread 항목에 해당 조건을 기입한다.
      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(Alternative Thread에 해당하는 속성의 경우 속성 이름 앞에 'A+number' 형식의 키워드 기술).
2. 단계 1을 수행하여 얻어진 컴포넌트 속성 테이블의 각 행에 대하여 다음 작업 반복을 통하여 Component State 항목을 완성한다.
   2.1 Attribute 항목에 기술된 해당 컴포넌트의 속성 및 가능한 속성 값의 분석을 통하여 컴포넌트의 상태이름을 정의한다(Alternative Thread에 해당하는 속성 값의 변화는 고려하지 않음).
   2.2 더 이상 추가될 상태가 없을 때까지 2.1을 반복한다.

3.1.3 상태식별테이블 작성

컴포넌트 속성식별테이블 하나당 하나의 상태식별테이블을 작성한다. 컴포넌트 속성식별테이블에 기술되어진 각 컴포넌트들의 속성들을 기반으로 가능한 속성 값들의 조합을 통하여 시스템의 상태를 식별한다. 상태의 식별은 컴포넌트 속성식별테이블의 Step No.순으로 각 컴포넌트의 가능한 속성 값들의 조합을 찾아 상태들을 식별해 나간다.

상태식별테이블의 각 항목에는 표 4와 같은 내용을 기입한다.

상태식별테이블의 작성 과정은 다음과 같다.

상태식별테이블 작성 규칙

Input : Components Attributes Identification Table3
Output : State Identification Table

1. 컴포넌트 및 해당 컴포넌트들의 가능한 속성들을 Comp. ID. 항목과 Attributes 항목에 기입한다 (Alternative Thread에 해당하는 속성 값은 상태와 무관하기 때문에 해당 속성으로 고려하지 않음)
2. 컴포넌트 속성식별테이블의 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 Thread 가 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.

3.1.4 EFSM 명세 생성

3.1.1-3.1.3은 EFSM 명세에 필요한 정보들을 유도하기 위한 과정이다. EFSM 명세의 각 전이들의 형식은 다음과 같다.

From-State
{Input} {Output} {Predicates} {Actions} {Color} To-State

From-State에서 To-State로의 전이는 시스템이 취하는 하나의 행위를 의미한다. 특히 상호운용성 테스트는 통합된 시스템들이 상호운용할 때 발생하는 실패들에만 관심을 가지므로[6] 2장에서 기술한 하나의 전이를 구성하는 요소들 외에도 지역 활동과 상호운용 활동을 구분하는 컬러의 항목을 추가하여 구성한다. 앞서 정의한 SFS와 컴포넌트 속성식별테이블, 상태식별테이블들의 각 항목들은 EFSM 명세를 작성할 때 다음과 같이 사용된다.

- State : 상태식별테이블의 State
- Input : SFS의 Input
- Output : SFS의 Output
- Predicates : 속성식별테이블의 Alternative Thread
- Action : 속성식별테이블의 Attribute Values의 변화

상태식별테이블에서 얻어진 각 상태정보와 SFS 및

표 4 상태식별테이블 구성 항목

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

컴포넌트 속성식별테이블에서 얻어진 상태들 간의 전이에 필요한 정보를 이용하여 EFSM을 구성함으로써 시스템의 행위를 모델링한다. EFSM은 XML 형식으로 기술한다. XML은 어떤 플랫폼에서나 가독성 있는 포맷을 제공하기 때문에 테스트케이스 생성 도구의 입력으로 활용하기 용이하다. EFSM 명세의 XML DTD는 다음과 같다.

```
<!-- ======= Start Entity Declaration ======= -->
<!-- ======= End Entity Declaration ======= -->

<!-- ======= Start Element Declaration ======= -->
<!ELEMENT SFS (transition)+>
<!ELEMENT transition (from , trans_info , to)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT trans_info (input+ , output+ , action+ , predicate+,
color+)>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT predicate (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!-- ======= End Element Declaration ======= -->

<!-- ======= Start Attribute Declaration ======= -->
<!ATTLIST SFS       sid ID #REQUIRED>
<!ATTLIST transition id ID #REQUIRED>
<!-- ======= End Attribute Declaration ======= -->
```

3.2 테스트케이스 자동 생성도구

테스트케이스 자동 생성 도구는 본 연구에서 제안한 EFSM을 입력으로 하고, 테스트 케이스 자동생성을 위한 이론 및 알고리즘 적용을 시도해 보기 위하여 최소한의 기능만을 갖춘 프로토타입 형태로 구현하였다. 본 도구는 3.1절에서 기술한 EFSM 명세 생성 과정을 통해 생성된 EFSM 명세를 입력으로 하고, 2.2절에서 기술한 이론 및 알고리즘을 이용하여 테스트케이스들을 생성한다. 도구의 실제 실행 모습은 그림 13에 표현되어 있다.

그림 6은 본 연구에서 구현한 테스트케이스 자동 생성 도구의 유스케이스 다이어그램이다.

3.2.1 테스트케이스 생성

EFSM 명세를 토대로 상태전이그래프인 도달성 그래프(reachability graph)를 작성한다. 노드가 되는 상태(state)와 간선이 되는 전이(transition)를 추출하여 상태전이그래프를 생성한다. 그런 다음 상태전이그래프에서 강하게 연결된 요소(SCC) 및 단순 사이클 등을 검색하여 비순환 경로를 갖는 그래프인 DAG를 생성한다. 생성된 DAG에 선택한 Adequate-Coverage 알고리즘의 중복성 제거 규칙을 적용하여 불필요한 노드와 간선을 삭제하여 가능한 전체 비순환 경로를 생성한다. 이러한 과정을 거쳐 생성한 비순환 경로와 단순 사이클을 합성하여 본래 상태전이그래프의 노드들로 대체하면 일

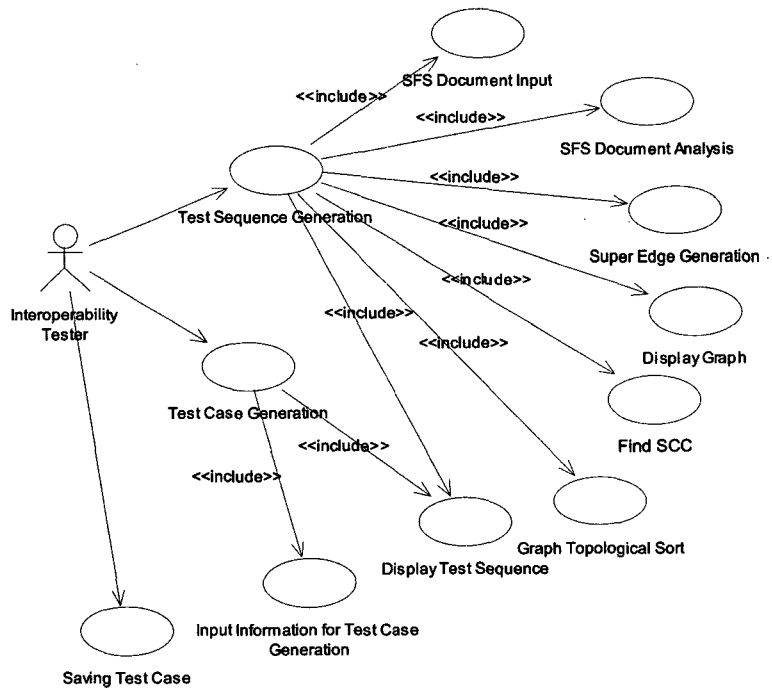


그림 6 Interoperability Test Generator의 유스케이스 다이어그램

련의 테스트 시퀀스들이 생성되게 된다.

3.2.2 테스트케이스 저장

3.2.1절의 과정을 거쳐 생성된 테스트케이스들은 미리 정의한 XML DTD에 따라 XML 문서로 생성되어 저장된다. XML 문서의 이름은 다음과 같은 형식으로 구성된다.

SFSNo_TestSuitNo_TestCaseNo

첫 번째 수는 SFS 문서의 번호를 의미하고 두 번째 수는 테스트슈트 번호, 그리고 마지막 수는 테스트케이스 번호를 의미한다.

테스트슈트들을 저장하기 위하여 정의한 XML DTD는 다음과 같다.

```
<!-- ======= Start Entity Declaration ======= -->
<!-- ======= End Entity Declaration ======= -->

<!-- ======= Start Element Declaration ======= -->
<!ELEMENT TestSuit (TestCase)+>
<!ELEMENT TestCase (Input+, output+, action+, predicate+)>
<!ELEMENT Input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT predicate (#PCDATA)>
<!-- ======= End Element Declaration ======= -->

<!-- ======= Start Attribute Declaration ======= -->
<!ATTLIST TestSuit
        SFSID ID #REQUIRED>
<!ATTLIST TestCase
        id ID #REQUIRED>
<!-- ======= End Attribute Declaration ======= -->
```

4. 테스트케이스 생성기법의 적용

본 장에서는 3장에서 기술한 테스트케이스 생성 기법을 실제로 시스템에 적용하여 본다. 먼저 적용할 시스템의 도메인을 설명하고 앞서 기술한 바에 따라 SFS, 컴포넌트 식별테이블, 상태식별테이블을 작성한 후 EFSM 명세서를 작성한다. 그리고 작성된 EFSM 명세를 입력으로 하여 테스트케이스 자동생성 도구에서 테스트케이스를 생성해내는 과정을 기술한다.

4.1 적용 도메인

본 논문에서 제안한 테스트케이스 생성기법을 적용할 예는 PNS(Personal Navigation System)를 이용한 대중교통 솔루션 시스템이다. 그림 7은 PNS를 이용한 대중교통 솔루션 중 '근접 대중교통 수단 제공' 기능을 보여주고 있다. 이 기능은 사용자가 휴대폰을 이용하여 어떤 목적지를 가기 위해 현재위치에서 가장 근접한 대중교통 수단을 검색하는 서비스이다. 이러한 시스템 각각의 기능은 유스케이스 모델에서 하나의 유스케이스로 매핑된다.

4.2 EFSM 명세 생성

본 절에서는 위치 기반 서비스 예제인 '근접 대중교통 수단 제공' 서비스를 3장에서 설명한 EFSM 모델링 기법을 적용하여 SFS, 속성식별테이블, 상태식별테이블을 작성하여 시스템의 가능한 상태를 추출한다.

대중교통 솔루션 시스템에 대한 시스템 분석서의 유스케이스, 유스케이스 명세서를 이용하여 상호운용과 관련한 시스템의 주요 구성요소를 식별한다. 식별된 구성

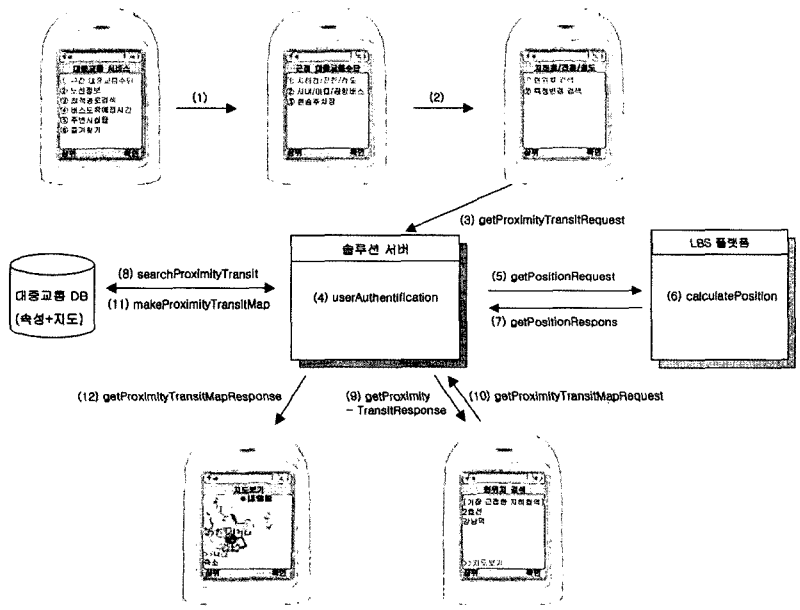


그림 7 '근접 대중교통 수단 제공' 서비스 과정

요소는 다음과 같다.

- T(Terminal) : 단말기
- TSP(Telematics Service Provider) : 텔레매틱스 서비스 제공자
- LBSP(Location Based Service Platform) : LBS 플랫폼
- CP(Contents Provider) : 컨텐츠 제공자

시스템 구성요소를 사용자의 단말(T), 솔루션 서버(TSP), 대중교통 DB(CP)로 분석하였다. 식별한 구성요소들 사이의 상호운용 메시지들을 파악하여 시퀀스 다이어그램을 작성한다. 그림 8은 사용자가 휴대폰을 사용하여 현재 위치에서 가장 가까운 대중교통 수단을 검색하는 서비스인 '근접 대중교통 수단 제공' 서비스 과정을 보이는 시퀀스 다이어그램이다.

그림 8의 '근접 대중교통 수단 제공' 시퀀스 다이어그램을 사용하여 3장의 1절에서 기술한 작성 규칙들을 적용한다. EFSM 명세 생성은 SFS, 속성식별테이블, 상태식별테이블 작성 단계를 통해 진행된다.

유스케이스 다이어그램, 유스케이스 명세서, 시퀀스 다이어그램을 종합적으로 분석하여 3.1.1절의 작성 규칙에 따라 SFS를 작성한다. 이 SFS는 '근접 대중교통 수단 제공'을 위한 SFS로서 '근접 대중교통 수단 제공'이라는 시스템 행위를 명확하게 단계적으로 세분하고, 각 단계의 행위와 관련된 데이타를 나타낸다. 그 결과 14단계의 컴포넌트 행위를 식별하였고, 그 각 행위에 사전조건, 입력, 사후조건, 출력에 관한 정보를 얻었다(표 5).

3.1.2와 3.1.3절에 기술된 바에 따라 PNS를 이용한 대중교통 솔루션 시스템의 '근접 대중교통 수단 제공' 서비스에 대한 속성식별테이블, 상태식별테이블을 작성한 결과는 표 6, 표 7과 같다. 이것을 기초로 EFSM 명세를 생성하고 도구의 입력으로 사용하기 위하여 정의한 XML 문서는 그림 9와 같다.

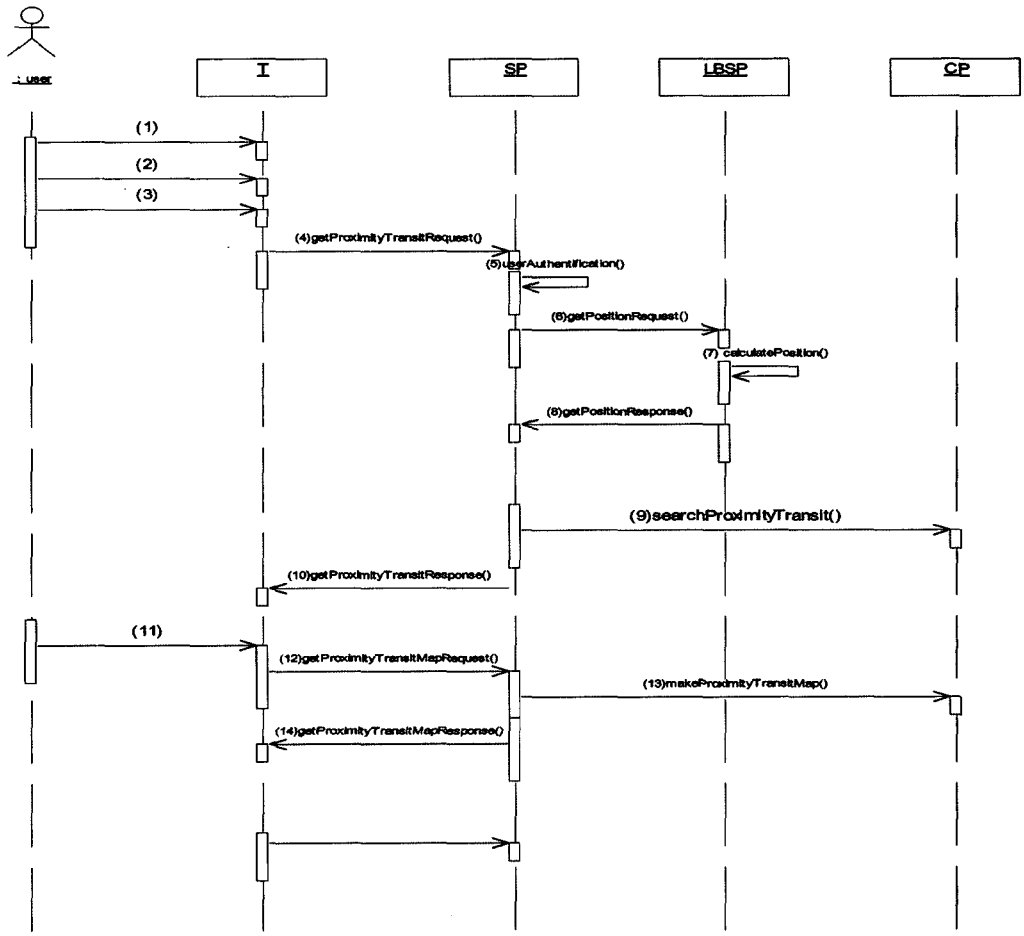


그림 8 '근접 대중교통 수단 제공'의 시퀀스 다이어그램

표 5 '근접 대중교통 수단 제공'서비스의 SFS

Function No. 1 ProximityPublicTransit						
Step	Component ID	PreCondition	Input	Behavior	PostCondition	Output
1	T	on 'publicTransitService' menu	select 'searchProximityTransit'	T displays 'searchProximityTransit' menu	on 'searchProximityTransit' menu	display menu
2	T	on 'searchProximityTransit' menu	select 'aTransit'	T displays the seleted transit menu	on the selected transit menu	display menu
3	T(TSP)	on the selected transit menu	select 'getProximityTransitRequest'	T requests TSP to get proximity transit	Timeout error	Timeout TSP
					TSP got 'getProximityTransitRequest'	
4	TSP	receive requesting 'getProximityTransitRequest'	cell phone number	TSP authenticates user	T got unregistered user error	Unregistered user error
					compete user authentication	
			.....			
14	TSP(T)	success making map		TSP sends proximity transit map to T	T displays proximity transit map	display map

표 6 '근접 대중교통 수단 제공'서비스의 속성식별테이블

Step No.	Component	Behaviors	Alternative Thread	Attributes	Component State
1	T	display 'searchProximityTransit' menu		idle_T {T, F} dispProxTransitMenu_T {T, F}	T : idle_T, F : wake_T T : dispProxTransitMenu_T, F :
2	T	display the seleted transit menu		dispSelectedTransitMenu_T {T, F}	T : dispSelectedTransitMenu_T, F:
3	T TSP	request TSP to get proximity transit(T)	TSP got Request/Fail	reqProxTransitToTSP_T {T, F} idle_TSP {T, F} getReqProxTransitFromT_TSP {T, F}	T : reqProxTransitToTSP_T, F : T : idle_TSP, F : wake_TSP T :getReqProxTransitFromT_TSP, F :TimeoutErrTSP_T
4	TSP	authenticate user(TSP)	Authenticate Success/Fail	authUserSucc_TSP {T, F}	T :authUserSucc_TSP, F : authUserFail_TSP
				.....	
14	TSP T	sends proximity transit map to T(TSP)		sendProxTransitMapToT_TSP {T, F} getProxTransitMapFromTSP_T {T, F}	T : sendProxTransitMapToT_TSP, F : T : getProxTransitMapFromTSP_T, F :

표 7 '근접 대중교통 수단 제공'서비스의 상태식별테이블

Comp. ID.	Attributes	Attribute Values																					
T	idle_T	T	F																				
	dispProxTransitMenu_T		T																				
	dispSelectedTransitMenu_T			T																			
	reqProxTransitToTSP_T				T																		
	getProxTransitFromTSP_T																	T					
	TimeoutErrTSP_T				A1																		
					T	F																	
	dispProxTransit_T																T			T			
	reqSearchProxTransitMapToTSP_T																				T		
	getProxTransitMapFromTSP_T																						T
.....																							
LBSP	idle_LBSP	T							F						T								
	getReqPosInfoFromTSP_LBSP								T														
	calPosSucc_LBSP										A4												
											F	T											
	sendPosInfoToTSP_LBSP											T											
CP	Idle_CP	T													F				F	T			
State No.		S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S10	S18	S19	S20	S21	S22

그림 9 '근접 대중교통 수단 검색' 서비스의 EFSM 명세에 대한 XML 문서

```
<?xml version="1.0"?>
<SFS sid = "01">
        <transition id = "S1">
                <from>Idle_TIdle_TSPIdle_LBSPIdle_CP
                </from>
                <trans_info>
                        <input>serarchProximityTransit
                        </input>
                        <output>dispMenu
                        </output>
                        <predicates>
                        </predicates>
                        <actions>
                                <act1>Idle_T=false
                                </act1>
<act2>dispProxTransitMenu_T=true
                                </act2>
                                <act3>
                                </act3>
                        </actions>
                        <color>White
                        </color>
                </trans_info>
                <to>wake_TdispProxTransitMenu_T</to>
        </transition>
        <transition id = "S2">
                <from>wake_TdispProxTransitMenu_T
                </from>
                <trans_info>
                        <input>aTransit
                        </input>

                        .....
```

4.4 개방형 서버 솔루션의 테스트 시퀀스 생성

작성된 EFSM 명세에 따라 PNS를 이용한 대중교통 솔루션 시스템이 제공하는 서비스 중 하나인 '근접 대중교통 수단 제공'에 대한 상태전이그래프 G는 그림 10과 같다.

그림 11은 Adequate-Coverage 알고리즘에 따라 화이트, 블랙, 슈퍼 간선을 표시한 그래프이다.

그림 10의 상태전이그래프 G에 Adequate-Coverage 알고리즘의 1행부터 8행을 수행한 결과 그래프는 그림 12와 같다.

다음으로 PATHS-IN-DAG 알고리즘에 따라 비순환 경로를 생성하기로 한다. 먼저 그림 10의 DAG를 위상정렬한 결과인 G' = {S1, S3, S5, SCC, S9, S12, S14,

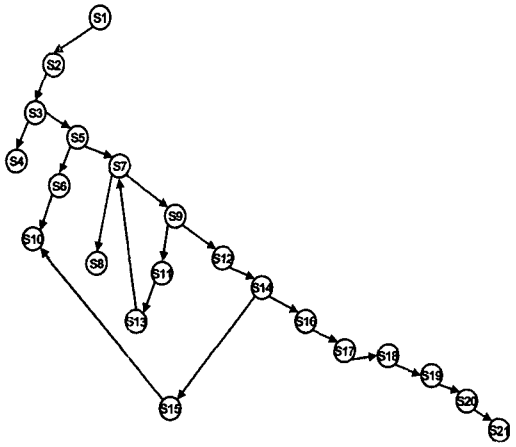


그림 10 '근접 대중교통 수단 제공' 서비스의 상태전이 그래프

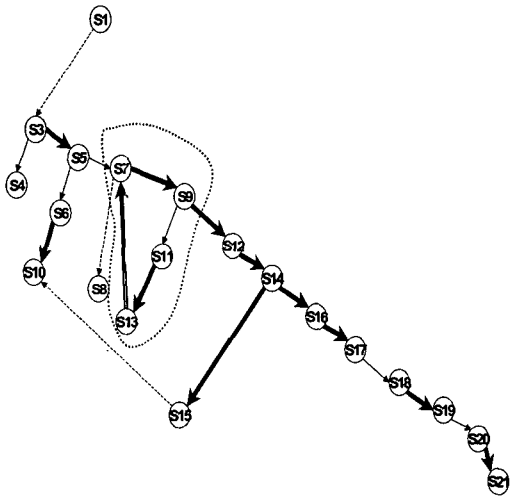


그림 11 White-Edge, Black-Edge, Super-Edge를 표시한 도달성 그래프

S16, S17, S18, S19, S20, S21, S15, S8, S6, S10, S4)이다. PATHS-IN-DAG 알고리즘의 2행에서 7행을 수행하면 비순환 경로 집합 P'는 다음과 같다.

> P' = {(S1, S3, S4), (S1, S3, S5, S6, S10), (S1, S3, S5, SCC, S8), (S1, S3, S5, SCC, S9, S12, S14, S16, S17, S18, S19, S20, S21), (S1, S3, S5, SCC, S12, S14, S15, S10)}

그러나 (S1, S3, S4)는 Adequate-Coverage 알고리즘의 9행에 있는 슈퍼 간선은 화이트 간선이나 다른 슈퍼 간선 바로 앞이나 뒤에 오지 않아야 한다는 조건에 위배되므로 제외시키면 P'는 다음과 같아지게 된다.

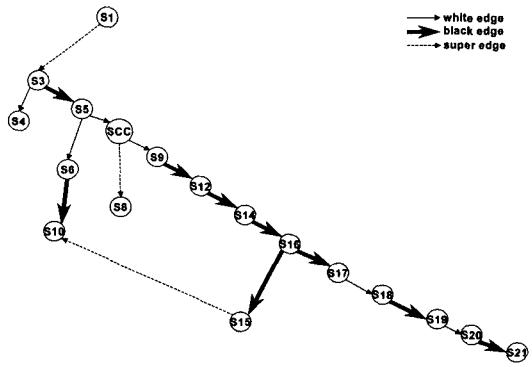


그림 12 그림 11에 대한 DAG

> P' = {(S1, S3, S5, S6, S10), (S1, S3, S5, SCC, S8), (S1, S3, S5, SCC, S9, S12, S14, S16, S17, S18, S19, S20, S21), (S1, S3, S5, SCC, S12, S14, S15, S10)}

P'에서 SCC를 본래의 상태전이그래프 G의 노드로 대체하면 P는 다음과 같은 형태가 된다.

> P = {(S1, S3, S5, S6, S10), (S1, S3, S5, S7, S8), (S1, S3, S5, S7, S9, S12, S14, S16, S17, S18, S19, S20, S21), (S1, S3, S5, S7, S12, S14, S15, S10)}

SCC에서 단순 사이클은 (S7, S9, S11, S13)이므로 P와 합성하면 생성된 테스트 시퀀스는 다음과 같다.

> T = {(S1, S3, S5, S6, S10), (S1, S3, S5, S7, S9, S11, S13, S7, S8), (S1, S3, S5, S7, S9, S11, S13, S7, S9, S12, S14, S16, S17, S18, S19, S20, S21), (S1, S3, S5, S7, S9, S11, S13, S7, S12, S14, S16, S15, S10)}

테스트 시퀀스 T에 따라 Adequate-Coverage 알고리즘을 적용한 결과 생성된 테스트케이스는 표 8과 같다.

테스트슈트 1의 경우에 해당하는 테스트 시나리오를 기술하면 다음과 같다.

(1) 구성요소 T에서 'searchProximityTransit' 메뉴를 선택하면 선택된 메뉴를 T에 표시하고 (2) 구성요소 T에서 'getProximityTransitRequest' 메뉴를 선택하면 (3) 구성요소 TSP는 구성요소 T의 전화번호를 입력으로 사용자 인증을 수행하되 (4) 인증에 실패하면 구성요소 T에 에러메시지를 표시한다.

테스트슈트 1은 구성요소 T와 TSP 사이의 상호운용성을 테스트하는 테스트케이스들이다.

4.5 테스트케이스 생성 도구

테스트케이스 생성 도구는 EFSM을 명세한 XML 문서를 입력으로 하여 앞서 기술한 이론 및 알고리즘들을 적용하여 테스트케이스를 자동 생성한다.

그림 13은 입력된 XML 문서로부터 상태와 전이 정

표 8 '근접 대중교통 수단 제공' 서비스의 테스트케이스

번호	테스트케이스
1	searchProximityTransit/dispMenu, getProximityTransitRequest/, cellPhoneNum/, /dispErrMsg
2	searchProximityTransit/dispMenu, getProximityTransitRequest/, cellPhoneNum/, cellPhoneID/, /NAK, NAK/, cellPhoneID/errMsg
3	searchProximityTransit/dispMenu, getProximityTransitRequest/, cellPhoneNum/, cellPhoneID/, /NAK, NAK/, cellPhoneID/, /positionInfo, positionInfo/, positionInfo/, /dispProxTransit, reqSearchProxTransitMap/, /dispTransitMap
4	searchProximityTransit/dispMenu, getProximityTransitRequest/, cellPhoneNum/, cellPhoneID/, /NAK, NAK/, cellPhoneID/, /positionInfo, positionInfo/, positionInfo/, /dispErrMsg

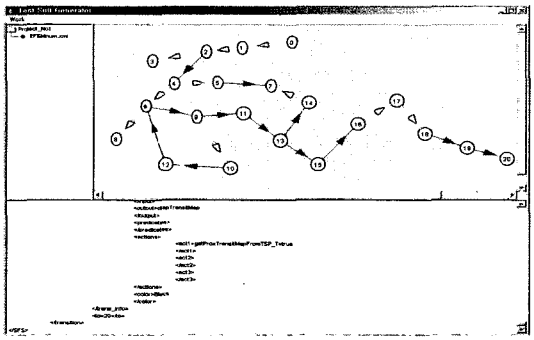


그림 13 상태전이그래프 구성한 결과

보를 추출하여 생성한 상태전이그래프이다. 그림 11에서 머리가 빈 실선 화살표는 화이트 간선을, 머리가 채워진 실선 화살표는 블랙 간선을 의미한다.

그림 14는 'Create SCC' 메뉴를 선택하여 SCC와 단순 사이클을 찾아 SCC로 대체하여 생성한 DAG를 보여주고 있다. 그림 14에서 SCC는 단순 사이클을 대체한 노드이다.

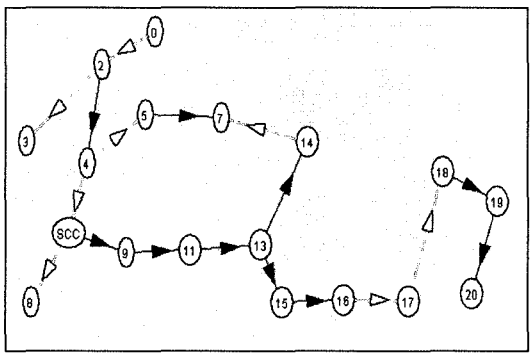


그림 14 그림 13의 상태전이그래프에 대한 DAG

5. 토 의

본 논문에서는 비정형적인 형식으로 기술된 문서로부터 EFSM 명세를 생성하는 기법과 EFSM 명세를 이용하여 테스트케이스를 생성하는 프로토타입을 구현하는 연구 결과를 기술하였다. 기존 연구들은 EFSM 명세로부터 상호운용성 테스트를 위한 테스트케이스 생성 기법들에 관한 것이거나[1,4-8] 테스트케이스 생성을 위한 알고리즘에 관한 연구[2,3]들이 대부분이다. 정형화된 문서로부터 EFSM 명세를 생성하는 것은 어려운 일이 아닐 수 있지만 비정형적 형식으로 기술된 문서로부터 필요한 정보들을 찾아내야하는 경우에는 숙련된 기술과 많은 경험을 필요로 한다. 본 논문에서는 이러한 경우에 적용할 수 있도록 EFSM 명세 생성 기법을 정의하였으며 그 결과를 이용하여 테스트케이스 생성 프로토타입을 구현하였다. 본 논문의 의의는 다음과 같다.

- 비정형적 형식으로 기술된 문서로부터 EFSM 명세 생성 기법 정의
  - 행위 정보 기술을 위한 구조화된 문서 형식 정의
  - 속성 및 상태 식별 방법 정의
  - PNS 시스템 사례 적용
- 상호운용성 테스트케이스 생성을 위한 프로토타입 구현
  - 정의한 기법에 따라 생성된 EFSM 명세를 이용한 테스트케이스 생성기의 프로토타입 구현

사실 비정형적 문서를 정형화하는 것은 불가능할 수도 있는 일이다. 논문에서는 수동적으로 EFSM 명세를 생성하고 있기 때문에 테스트케이스 생성을 위한 전과정을 자동화하지 못하고 일부만 자동화할 수 있었다. 향후 추가적인 연구가 필요한 부분은 다음과 같다.

- 수동성이 갖는 오류를 최소화하기 위한 추가적인 행위 정보 추출 규칙 정의
- 다양한 시스템을 통한 EFSM 명세 생성 기법의 검증
- 완전한 상호운용성 테스트케이스 생성기 구현

6. 결론 및 향후 연구

본 논문에서 기술한 테스트케이스 생성 기법 및 도구의 프로토타입은 개방형 서버의 상호운용성뿐만 아니라 독립적으로 구축된 시스템들 사이의 상호운용성을 테스트하기 위한 기본 자료로서 활용될 수 있다. 왜냐하면 도메인이 달라지더라도 상호운용하는 행위의 주체와 제

어나 데이타의 흐름에 대한 정의만이 달라질 뿐 주요 속성 및 상태추출기법, EFSM 모델을 이용한 도달성 그래프 생성기법, 테스트케이스 생성 기법은 동일하게 적용될 수 있기 때문이다. 또한 테스트의 커버리지나 시스템의 중요도에 따라 이미 적용되어 사용된 알고리즘을 변형한다면 테스트 목적에 맞는 최적의 테스트케이스를 생성할 수 있다.

본 논문에서 도출한 개방형 서버의 상호운용성 테스트를 위한 테스트케이스와 도구의 프로토타입은 개방형 서버를 이용한 솔루션 시스템 구축시 필수적인 상호운용성 테스트에 있어서 완전성 및 중복성이 배제된 상호운용성 테스트를 가능하게 함으로써 프로덕트의 품질 및 테스트에 소요되는 비용의 최소화에 기여할 것이다. 또한 테스트케이스 생성 기법을 통해 생성된 테스트케이스는 DB와 같은 저장장소에 저장된 후 상호운용성 테스트 자동화에 활용될 수 있다.

그러나 문헌 [9,10]과 같이 요구사항 문서로부터 테스트케이스의 생성에 이르기까지의 과정을 최대한 자동화하여 수동성이 갖는 오류를 최소화하기 위하여 보다 상세한 알고리즘이나 규칙에 대한 정의가 필요하다고 본다.

참 고 문 헌

[1] Glenn, R., Frankel, S., Montgomery, D., "The NIST IPsec Web-Based Interoperability Test System," Proceedings of IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 147-152, 2000.

[2] Moseley, S., Randall, S., Wiles, A., "Experience Within ETSI of the Combined Roles of Conformance Testing and Interoperability Testing," The 3rd Conference on Standardization and Innovation in Information Technology, pp. 177-189, 2003.

[3] Qixiang, P., Shiduan, C., Yuehui, J., "Protocol Conformance Test Suite Generation," Proceedings of International Conference on Communication Technology, pp. 218-222, 1996.

[4] Wang, C. -J., Liu, M. T., "Generating Test Cases for EFSM with Given Fault Models," Proceedings of 12th Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 774-781, 1993.

[5] Petrenko, A., Boroday, S., Groz, R., "Confirming Configurations in EFSM Testing," IEEE Transactions on Software Engineering, Vol. 30, Issue 1, pp. 29-42, 2004.

[6] Lee, D., Yannakakis, M., "Principles and Methods of Testing Finite State Machines-A Survey," Proceedings of the IEEE, Vol. 84, Issue 8, pp. 1090-1123, 1996.

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

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

[9] Malek, M., Dibuz, S., "Pragmatic Method for Interoperability Test Suite Derivation," Proceedings of 24th Euromicro Conference, Vol. 2, pp. 838-844, 1998.

[10] Tsai, Bor-Yuan, Stobart, S. Parrington, Mitchell, N., "An Automatic Test Case Generator Derived from State-Based Testing," Asia Pacific Software Engineering Conference, pp. 270-277, 1998.

이 지 현

1993년 전북대학교 정보통신공학과 졸업 2000년 전북대학교 교육대학원 전자계산교육(교육학석사). 2005년 전북대학교 대학원 전산통계학과(이학박사). 2005년~현재 한국정보통신대학교 소프트웨어공학연구소 선임연구원. 관심분야는 정형명세, 테스팅, 임베디드 시스템 아키텍처, PMM

노 혜 민

2000년 전북대학교 컴퓨터과학과 졸업(이학사). 2002년 전북대학교 대학원 전산통계학과 졸업(이학석사). 2002년~현재 전북대학교 대학원 컴퓨터통계정보학과 박사과정. 관심분야는 컴포넌트 기반 소프트웨어 개발, 정형 명세 기법, 소프트웨어 테스팅, LBS

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 자연과학대학 컴퓨터과학과 전임강사~부교수. 관심분야는 소프트웨어 개발 프로세스, 소프트웨어 품질, 컴포넌트 소프트웨어, 소프트웨어 메트릭스, 소프트웨어 에이전트, GNSS, GIS, 교육공학, 인지과학 등

장 옥 배

1973년 고려대학교 졸업(학사,석사). 1988년 산타바바라대 대학원(Ph. D.) 1974년~1980년 죠지아 주립대. 오하이오 주립대 박사과정 수료. 1980년~현재 전북대학교 자연과학대학 전자정보공학부 교수. 관심분야는 소프트웨어공학, 전산교육, 수치해석, 인공지능 등

이 준 욱

1997년 충북대학교 컴퓨터과학과 졸업. 1999년 충북대학교 대학원 전자계산학과(이학석사). 2003년 충북대학교 대학원 전자계산학과(이학박사). 2003~2004 ETRI 텔레매틱스연구단 PostDoc. 연구원. 2004년~ 현재 ETRI 텔레매틱스. USN연구단 선임연구원. 관심분야는 시공간 데이타마이닝, 위치기반 서비스, 상황인식, 텔레매틱스