DOI: 10.3745/KIPSTD.2010.17D.2.147

테스트 수행시간을 고려한 임베디드 소프트웨어의 적합성 테스트 시나리오 추출 기법

박 인 수[†] · 신 영 술[††] · 안 성 호[†††] · 김 진 삼[††††] · 김 재 영[†††††] · 이 우 진[††††††]

요 약

임베디드 소프트웨어의 적합성 테스트는 소프트웨어의 기능이 명세를 따라 정확히 구현되었는지 검사하는 것이다. 적합성 테스트에서 테스트 시나리오는 소프트웨어의 전체 기능을 테스트할 수 있도록 추출되어야 한다. 일반적으로 테스트 시나리오는 단순히 전체의 기능들을 한 번씩 테스트해 보는데 초점이 맞춰져 있다. 하지만 테스트 시나리오는 테스트 수행의 효율성을 고려할 필요가 있다. 이 연구에서는 각 함수들을 테스트하는데 걸리는 시간과 사용자 입력으로 인해 발생하는 대기 시간을 고려하여 최적화된 테스트 시나리오를 추출하는 기법을 제안한다. 테스트 시나리오를 추출하기 위해 상태 머신 다이어그램과 테스트 케이스를 바탕으로 그래프 형태의 테스트 모델을 생성한다. 테스트 모델에는 테스트 수행 시간과 사용자 입력에 대한 정보가 포함되어 있다. 최적화된 테스트 시나리오는 테스트 모델을 기반으로 수정된 최단거리 알고리즘을 이용하여 추출한다. 제안하는 테스트 시나리오 작성 기법을 이용하면 테스트 수행 시간을 줄일 수 있고, 테스트 자동화를 향상시킬 수 있다.

키워드 : 테스트 시나리오, 적합성 테스트, 임베디드 소프트웨어

Conformance Test Scenario Extraction Techniques for Embedded Software using Test Execution Time

In Su Park[†] · Youngsul Shin[††] · SungHo Ahn[†††] · JinSam Kim[††††] · JaeYoung Kim[†††††] · Woo Jin Lee[††††††]

ABSTRACT

Conformance testing for embedded software is to check whether software was correctly implemented according to software specification or not. In conformance testing, test scenarios must be extracted to cover every test cases of software. In a general way, test scenarios simply focus on testing all functions at least one time. But, test scenarios are necessary to consider efficiency of test execution. In this paper, we propose a test scenario extraction method by considering function's execution time and waiting time for user interaction. A test model is a graph model which is generated from state machine diagram and test cases in software specification. The test model is augmented by describing test execution time and user interaction information. Based on the test model, test scenarios are extracted by a modified Dijkstra's algorithm. Our test scenario approach can reduce testing time and improve test automation.

Keywords : Test Scenario, Conformance Test, Embedded Software

1. 서 론

소프트웨어의 테스팅 방법 중 적합성 테스트는 구현된 소프트웨어가 표준 명세에 정의된 기능들을 정확히 구현하는지를 검사하는 것이다[1]. 일반적으로 오류 검출 테스트의 경우 소프트웨어 내부의 오류를 검사하는데 중점을 둔다. 이와 다르게 적합성 테스트는 구현된 소프트웨어가 표준 또는 규격을 만족하는지 여부와 주변 다른 모듈들과의 적합성에 중점을 둔다. 적합성 테스트의 대상은 소프트웨어에서 외부로 제공되는 인터페이스이고, 오류 검출 테스트는 외부와 내부의 모든 함수에 대해 테스트를 수행한다.

구현된 소프트웨어에 대해 적합성 테스트를 수행하기 위해서는 명세서에 기술되어 있는 소프트웨어의 기능들이 모두 테스트되어야 한다. 테스트 대상이 되는 기능은 해당하는 테스트 케이스를 수행하여 테스트할 수 있다. 소프트웨

※ 본 연구는 한국전자통신연구원 대경권연구센터의 위탁과제 지원으로 수행되었음.
† 정 회 원 : 삼성전자 무선사업부 사원
†† 준 회 원 : 경북대학교 전자전기컴퓨터학부 박사과정
††† 정 회 원 : 한국전자통신연구원 대경권연구센터 자동차SW플랫폼연구팀 선임연구원
†††† 정 회 원 : 한국전자통신연구원 대경권연구센터 자동차SW플랫폼연구팀 책임연구원
††††† 준 회 원 : 한국전자통신연구원 대경권연구센터 자동차SW플랫폼연구팀 팀장
†††††† 정 회 원 : 경북대학교 IT대학 컴퓨터학부 부교수
논문접수 : 2010년 2월 26일
수 정 일 : 1차 2010년 3월 31일
심사완료 : 2010년 3월 31일

어의 기능을 테스트하기 위해 선정된 테스트 케이스를 수행 순서가 있는 집합으로 묶어 놓은 것이 테스트 시나리오이다. 테스트 시나리오는 테스트 스윗 혹은 테스트 시퀀스라고도 하며 실제로 테스트를 수행하기 위한 절차를 나타낸다. 이러한 테스트 시나리오를 효율적으로 구성하는 것도 적합성 테스트를 수행하기 위해 중요한 과제이다.

특히 임베디드 소프트웨어 경우는 일반 소프트웨어와는 다르게 환경적인 요소와 시간 제약적인 요소에 영향을 많이 받는다[2]. 시스템의 특정 상태에 도달하는 여러 가지 경로가 존재할 수 있으며 특정 경로의 경우 사용자 입력이나 주변환경과의 복잡한 상호작용을 거쳐야 하는 상황이 생길 수도 있다. 즉, 테스트 케이스를 불필요하게 중복해서 수행하거나 비효율적인 경로로 테스트 케이스를 수행하게 되면 테스트 수행시간을 증가시키는 문제를 야기시킬 수 있다. 그러므로 임베디드 소프트웨어의 경우는 일반 소프트웨어와 달리 테스트의 효율성을 확보하기 위해서는 사용자 입력뿐만 아니라 단위 행위들의 수행시간을 고려하여 테스트 시나리오를 효율적으로 구성하여야 한다.

이 연구에서는 모든 테스트 케이스를 고려하되, 테스트 케이스를 수행하는 경로를 최적화하는 기법을 제안한다. 소프트웨어의 행위에 대한 명세와 생성된 테스트 케이스를 바탕으로 적합성 테스트를 수행하기 위한 테스트 시나리오를 추출한다. 일반적으로 소프트웨어의 행위에 대한 명세는 표준 문서, 개발자가 작성한 소프트웨어 명세서에 기술되어 있으며, 주로 UML의 상태 머신 다이어그램[3]으로 표현된다. 또한 표준 문서와 소프트웨어 명세서에는 테스트 케이스에 대한 내용도 기술되어 있다. 소프트웨어의 행위를 표현한 상태 머신 다이어그램과 분류 트리 방법을 통해 생성된 테스트 케이스를 정형화된 그래프의 형태로 변환하여 테스트 모델을 생성한다. 그리고 테스트 수행의 효율을 높이기 위해 소프트웨어의 함수에 대한 수행시간과 사용자 입력으로 인해 발생하는 대기 시간을 고려하여 최적화된 테스트 시나리오를 추출한다.

이 연구의 구성은 다음과 같다. 제 2 절에서 기존의 적합성 테스트와 테스트 시나리오 추출에 대한 관련 연구 내용을 기술하고, 제 3 절에서는 제안하는 테스트 시나리오 추출 기법에 대해 기술한다. 제 4 절에서 추출된 시나리오의 수행시간을 기존의 시나리오 추출 방법과 비교한 결과를 기술하고, 제 5 절에서는 결론을 기술한다.

2. 관련 연구

임베디드 소프트웨어는 타켓 시스템에 놓여지기 때문에 효율적인 테스트를 수행하기 위해서는 타켓 시스템에는 최소의 연계 모듈만 두고 호스트 환경에서 원격으로 테스트를 진행한다. 이 연구에서는 이러한 테스팅 환경을 가정하고 호스트 환경에서 원격으로 어떠한 순서로 타켓 시스템을 수행할 지를 나타내는 테스트 시나리오를 효율적으로 생성하는 것을 목표로 삼고 있다. 테스트 시나리오 생성에 관한 연구는 크게 시스템의 정형 명세 기반의 시나리오 생성 방법과 표준 API 기반 시나리오 생성 방법으로 나뉠 수 있다.

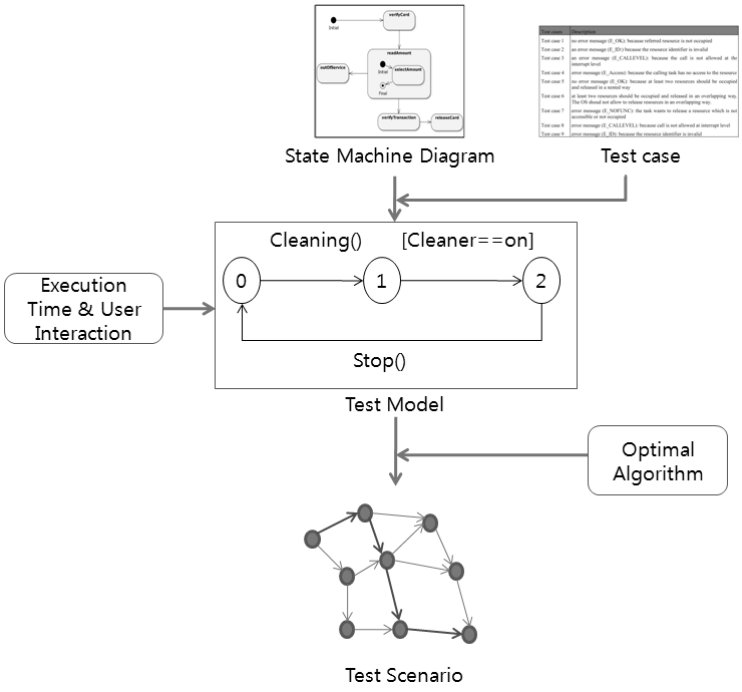
정형명세 기반의 시나리오 생성 기법에서는 우선적으로 시스템의 요구사항을 정형적으로 표현한다. 요구사항 정형 모델은 요구사항 명세를 정확하게 반영하며 시스템의 모든 행위를 기술하기 때문에, 정형 모델을 이용하여 테스트 케이스뿐만 아니라 테스트 시나리오도 생성할 수 있다[4-5]. Linzhang[6]은 Event-B를 이용하여 시스템과 테스트 시나리오를 작성하고, 실행 가능한 테스트 케이스의 템플릿을 생성한다. 하지만 정형 언어는 다루는데 어렵고, 크기가 큰 시스템의 모델링에는 어려움이 있다. UML의 액티비티 다이어그램을 이용하여 테스트 시나리오를 설계하고, 이로부터 각 시나리오별 테스트 케이스를 생성하는 연구가 존재한다[6-7]. UML에는 액티비티 다이어그램처럼 행위의 순서를 모델링할 수 있는 시퀀스 다이어그램이 제공되므로, 액티비티 다이어그램을 대신하여 시퀀스 다이어그램을 이용해서 테스트 시나리오를 생성할 수 있다[8]. 테스트 시나리오의 경로를 최적화하는 연구로는 시나리오를 특징짓는 방법과 시나리오를 겹치는 방법이 있다[9-10]. 하지만 테스트 시나리오 자체의 경로 수는 줄어들었으나, 테스트 시나리오를 구성하는 함수 수행에 드는 비용의 고려가 없으며, 내부 인터페이스와 외부 인터페이스의 구분이 없다.

표준 API 기반의 적합성 테스트 시나리오 생성에 관한 연구를 보면, 표준 문서에서 적합성 테스트를 위한 테스트 시나리오를 제공하고 있지만 테스트 시나리오를 설계하는 디자이너의 지식과 경험에 의존하며 테스트 시나리오 생성을 위한 체계적인 접근방법은 제공하지 않는다[11]. 이러한 접근 방법에서는 요구사항이나 표준 API를 바탕으로 테스트 케이스가 미리 정의되어 있으며 이들을 효율적으로 연동하는 것은 테스팅을 수행하는 사람의 몫이 된다. 이와 같이 테스트 케이스가 명시되어 있는 상황에서 테스트 케이스들을 효율적으로 연계하여 수행하기 위해서는 테스트 수행조건, 외부환경 요소, 사용자 입력, API의 파라미터 등을 분석하여 최적의 테스트 시나리오를 생성하는 것이 필요하다.

3. 테스트 시나리오 추출 기법

본 장에서는 수행시간과 사용자 입력에 최적화된 테스트 시나리오를 추출하기 위해 이 연구에서 제시하는 기법에 대해 기술한다. (그림 1)은 테스트 시나리오 추출 기법의 전체 프로세스를 나타낸다.

테스트 시나리오를 추출하기 위한 입력으로는 소프트웨어 행위를 표현한 명세와 테스트 케이스가 있다. 일반적으로 테스트 대상이 되는 소프트웨어의 행위는 유한 상태 머신(Finite State Machine)의 형태로 표현된다. 유한 상태 머신은 정형적 기술 기법의 기본적인 모델이다. 유한 상태 머신의 형태로 소프트웨어의 행위를 표현하는 여러 가지 기법들이 존재한다. SDL과 UML의 상태 머신 다이어그램이 이에 해당된다. 이 연구에서는 UML의 상태 머신 다이어그램으로



(그림 1) 테스트 시나리오 추출 프로세스

표현된 소프트웨어 행위를 분석하여 테스트 시나리오를 추출한다.

테스트 케이스는 소프트웨어의 행위를 기술하고 있는 명세서에 정의되어 있다. 일반적으로 테스트 케이스는 하나의 API를 테스트하기 위한 것으로 API를 실행하기 위한 상태, API의 이름, 파라미터와 출력 값을 기술한다. 하나의 API에 대한 테스트 케이스는 상태와 파라미터의 종류에 따라 여러 가지로 나누어질 수 있다.

이 연구에서는 상태 머신 다이어그램으로 표현된 소프트웨어의 행위를 간단한 가중치 그래프의 형태로 변환한다. 변환된 그래프를 분석하여 테스트 케이스와 일치하는 부분을 검사한다. 이렇게 생성된 그래프의 간선에 각 API의 수행시간과 사용자 입력 여부에 대한 가중치를 부여하여 테스트 모델을 생성한다. 생성된 테스트 모델은 평면적인 구조의 간단한 가중치 그래프이므로 최적화 알고리즘을 통해 모든 테스트 케이스를 수행하는 최단 거리의 경로를 추출할 수 있다. 이렇게 추출된 경로가 테스트 시나리오가 된다.

3.1 상태 기반 테스트 모델

3.1.1 상태 기반 테스트 모델 정의

앞에서 기술한 것처럼 소프트웨어의 행위는 일반적으로 UML의 상태 머신 다이어그램으로 표현된다. 그러나 상태 머신 다이어그램은 복합 구조, 계층 구조로 표현될 수 있기 때문에 상태 머신 다이어그램에서 테스트 시나리오를 추출하기는 쉽지 않고, 테스트 케이스와 동일한 부분을 찾아내어 매핑하기 힘들다. 이 연구에서는 소프트웨어의 행위를 나타내는 상태 머신 다이어그램을 간단한 그래프 형태의 테스트 모델로 변환하여 테스트 시나리오를 추출할 수 있도록 한다. 테스트 모델은 간단한 가중치 그래프의 형태로 다음과 같이 정의한다.

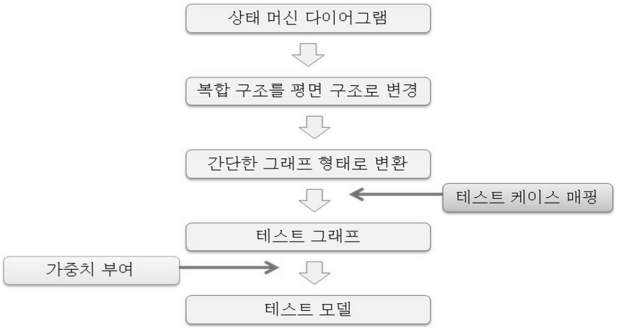
[정의 1] Test Model

Test Model(TM) = (S, A, { →S : s ∈ A }, S0)

- S : 상태들의 집합
- A : TM내에서 나타나는 행위집합
- →S : 모든 행위들의 시퀀스로부터 발생 가능한 전이 관계
- S0 : 초기상태

정의 1에서 A는 모든 행위를 뜻하는데, 행위에는 상태 머신의 경계 조건, 이벤트, 액션이 포함된다. 이 행위의 집합을 E라 한다. 또한 관찰 가능하지 않은 행위 “τ”도 포함된다. 즉, A = E ∪ {τ} 이다. 위와 같이 정의된 TM은 상태 머신의 계층적, 복합적 표현을 평면적으로 표현한다. 이렇게 간단한 TM으로 변환된 상태 머신을 이용하여 테스트 시나리오를 추출하고자 한다. 테스트 모델을 생성하기 위해서는 몇 가지 과정이 필요하다. (그림 2)는 테스트 모델 생성을 위한 흐름도이다.

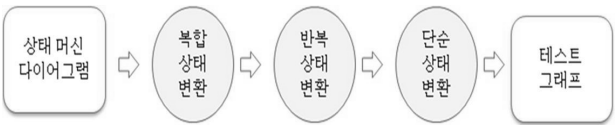
복잡한 구조의 상태 머신 다이어그램은 하나의 테스트 모델로 직접 변환될 수 없으며, 먼저 하나의 상태 머신에 존재하는 여러 영역에 대해 각각의 영역을 평면적인 구조로 변환한 후에 간단한 그래프로 변환한다. 이 과정에서 상태 머신의 변환에 대한 규칙이 필요하며, 이러한 규칙을 통해 테스트 케이스를 매핑하고 가중치를 부여할 수 있는 형태로 변환하여 테스트 모델을 생성한다.



(그림 2) 테스트 모델 생성 흐름도

3.1.2 상태 머신 다이어그램의 변환

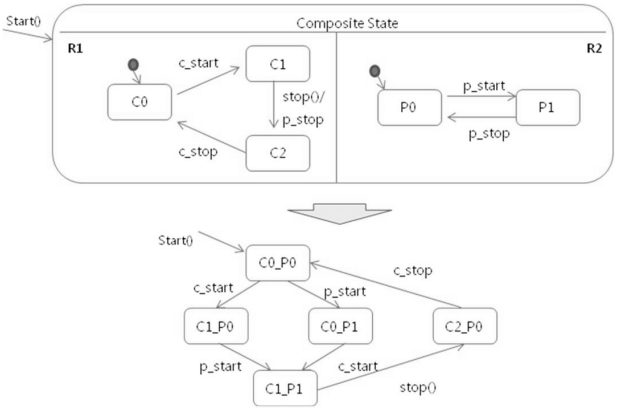
상태 머신 다이어그램을 단순 그래프로 변환하는 것은 입력 모델을 단순화하여 테스트 시나리오 추출을 효율적으로 진행하기 위함이다. 상태 머신과 테스트 모델 모두 상태와 전이를 가지는데, 상태 머신은 단순 상태, 복합상태 등을 가지는 반면, 테스트 모델은 단순 상태만을 갖는다. 또한 상태 머신은 경계 조건, 이벤트, 액션이 포함된 전이를 가지는 반면, 테스트 모델은 그 중 하나만이 표현된 전이를 갖는다. 따라서 상태 머신을 테스트 모델로 변환하기 위해서는 이러한 특징을 고려하여 다음과 같은 세가지 규칙을 갖는다. 변환 규칙은 (그림 3)의 상태 머신 변환 흐름도와 같이 진행된다. 먼저 복합상태들을 모두 검색하여 복합상태에 나타나 있는 계층 구조를 평면구조로 변환한다. 그리고 변수에 의한 반복 부분이 나타나는 부분은 반복상태 변환을 통해 변수를 제거한다. 마지막으로 전이의 경계 조건, 이벤트, 액션 부분들을 분할하여 단순 평면 그래프로 변환한다. 이 절에서는 복합상태 변환, 반복상태 변환, 단순상태 변환 기법을 하나씩 설명한다.



(그림 3) 상태 머신 변환 흐름도

· 복합 상태 변환

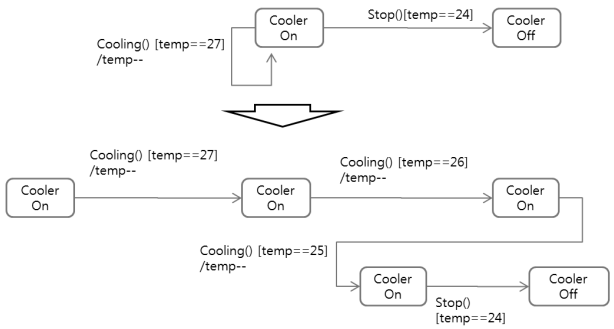
복합상태 변환은 상태의 내부에 계층적으로 다른 상태들을 포함하는 복합상태들의 내부 상태들을 한 단계 위로 올려 평면적인 구조로 변환하는 것을 말한다. 병행적인 상태들이 포함된 경우는 병행적인 상태들을 전체 순서 관점에서 하나의 상태도로 합성하여 나타내고 메시지 전송 이벤트들은 메시지 동기화 순서에 맞게 합성한다. (그림 4)는 두 개의 병행 영역을 가진 복합 상태를 평면적인 구조로 변환되는 모습을 나타낸다. 먼저, R1 영역의 c_start 전이와 R2 영역의 p_start 전이는 병행적으로 수행될 수 있으므로 변환된 모델에서와 같이 c_start → p_start 전이와 p_start → c_start 전이가 모두 나타난다. R1 영역의 C1 상태에서 발생하는 전이인 stop()/p_stop은 stop() 이벤트가 발생할 때, p_stop 메시지를 R2에 전송하는 것으로 이러한 메시지 전송이 발생하는 경우는 변환된 상태도에서와 같이 C1_P1 상태에서 동시에 C2_P0로 상태로 전이가 발생한다.



(그림 4) 복합 상태 변환

· 반복 상태 변환

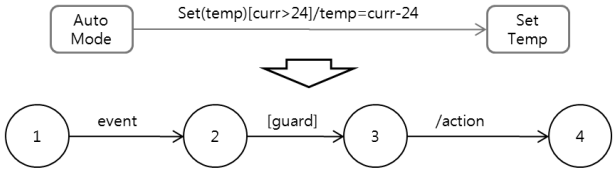
상태 머신의 반복 상태 변환은 변수의 사용으로 인해 다중의 의미를 내포하는 상태들을 나열하여 단순 의미를 가진 상태로 변환하고자 한다. 반복 상태의 경계 조건에 따라 반복의 횟수가 결정된다. 경계 조건이 없을 경우는 반복 상태를 한번만 수행하고 지나갈 수 있지만 반복 횟수가 정의되어 있을 때는 횟수만큼 반복을 수행해야 다음 상태로 전이할 수 있다. 반복 상태를 단순한 그래프로 만들 때는 반복 횟수에 따라 상태와 전이를 추가하여 반복을 순차적인 구조로 펼쳐야 한다. (그림 5)는 반복 상태 변환을 나타낸다. Cooling() 함수의 반복 횟수는 경계 조건을 비교해보면 3회라는 것을 알 수 있다. 그러므로 Cooler On 상태를 횟수만큼 추가하여 반복 상태를 순차 상태로 변환한다.



(그림 5) 반복 상태 변환

· 단순 상태 변환

상태 머신의 경우 (그림 6)와 같이 이벤트와 경계 조건, 액션이 포함된 복합 상태로 표현될 수 있다. 이와 같은 상



(그림 6) 단순 상태 변환

태를 단순하게 각각의 경계 조건과 이벤트, 액션을 분리하여 그래프로 변환한다.

3.1.3 테스트 케이스 매핑

테스트 케이스에는 테스트를 실행하기 위한 상태와 실행되는 API, 입력 파라미터 값, 결과 값이 나타난다. 테스트 케이스를 변환된 그래프에 매핑하기 위해서는 우선 실행 전 상태를 검사해야 한다. 이를 위해서 실제로 상태 머신 다이어그램을 간단한 그래프 형태로 변환하는 과정에서 경계 조건과 액션 등에 포함된 변수들을 수집하여 별도의 테이블에서 관리한다.

테스트 케이스 매핑 단계는 다음과 같다.

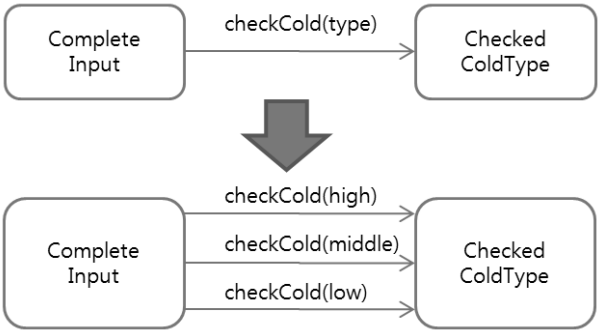
- 그래프의 처음 상태부터 그래프 전체를 순차적으로 순회한다.
- API 함수의 이름이 발견되면 테스트 케이스 테이블에서 AP함수와 이름이 같은 케이스를 찾는다.
- 테스트 케이스의 사전 조건과 그래프의 현재 상태에서의 변수 값을 비교한다.
- 그래프에서 다음 단계에 경계 조건이 존재하면 경계 조건이 현재 상태에서 만족되는지를 검사한다.
- 모든 조건이 만족되면 그 API 함수를 테스트 케이스로 표시한다.

(그림 7)은 테스트 케이스를 그래프에 매핑하는 부분이다.

- TC#1 : [mode==auto] Set(temp) [temperature]

또한, 같은 API함수의 테스트 케이스이면서 전 상태 또는 입력 파라미터가 다른 테스트 케이스들이 존재하면 테스트 수행 시에 모두 테스트되어야 하므로 그래프에 테스트 케이스의 수만큼 간선을 추가한다. (그림 8)은 checkCold() 함수의 입력 파라미터가 high, middle, low 일 때 테스트 케이스 간선을 추가하는 모습을 보여준다.

- TC#2 : checkCold(high)
- TC#3 : checkCold(middle)
- TC#4 : checkCold(low)



(그림 8) 테스트 케이스 간선 추가

테스트 케이스의 매핑까지 완료하게 되면 테스트 모델을 생성하기 위한 전 단계인 테스트 그래프가 생성된다.

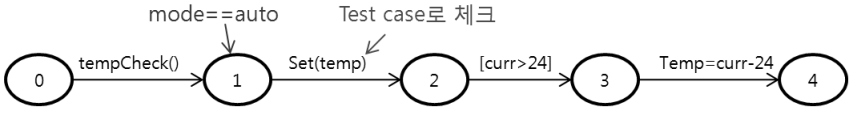
3.1.4 수행시간 및 사용자 입력에 대한 가중치 부여

테스트 그래프에서 하나의 간선은 하나의 함수 혹은 테스트 케이스를 의미한다. 각 함수는 테스트 실행 시에 수행되는 시간이 다르고, 어떤 함수는 사용자의 입력을 받아야 계속 진행이 가능하므로 테스트 실행은 함수의 수행에 걸리는 시간에 따라 전체 실행 시간에 대해 차이가 나타난다. 또한 사용자 입력이 있는 경우 테스터가 입력하여야 하기 때문에 테스트 자동화가 어렵고 입력에 따른 시간이 많이 걸린다.

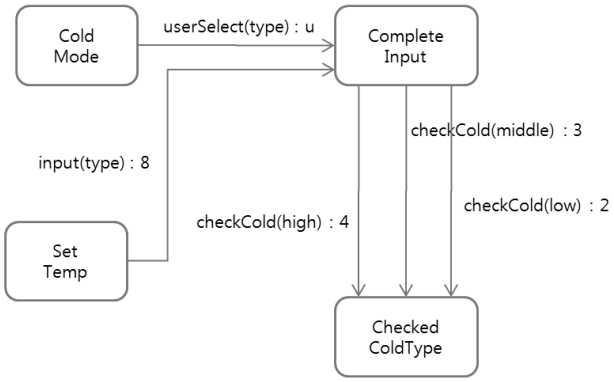
이러한 문제를 해결하기 위해서 테스트 모델의 각 간선에 가중치를 부여한다. 각 함수를 실행할 때 걸리는 시간은 개발 단계에서 검사가 가능하다. 각 함수의 수행시간에 대해 가중치를 부여함으로써 시간이 많이 걸리는 패스를 피할 수 있다. 또한 함수에 사용자 입력이 있는 경우에는 테스트 모델에서 그 부분에 해당하는 간선에 별도의 가중치(u)를 부여한다. 사용자 입력이 있는 패스를 피함으로써 테스트 자동화의 성능을 높일 수 있다. <표 1>은 API 함수의 수행시간 가중치 및 사용자 입력 가중치를 표로 나타낸 것이다. (그림 9)는 가중치 표를 참조하여 테스트 그래프의 간선에 수행시간 및 사용자 입력에 대한 가중치를 부여한 그림이다.

〈표 1〉 API 함수 가중치 표

function	Weight
input(type)	8
checkCold(high)	4
checkCold(middle)	3
checkCold(low)	2
userSelect(type)	u



(그림 7) 테스트 케이스 매핑



(그림 9) 가중치를 부여한 테스트 모델

3.1.5 테스트 모델의 예

시스템의 행위를 나타내는 상태 머신 다이어그램을 간단한 그래프의 형태로 변환한 후에 모든 테스트 케이스들과 일치하는 간선을 찾아 표시하고, 각 간선에 대한 수행시간 및 사용자 입력에 대한 가중치를 부여하면 테스트 모델이 완성된다. 완성된 테스트 모델은 간단한 가중치 그래프의 형태로 나타난다. (그림 10)은 생성이 완료된 테스트 모델을 보여준다.

3.2 테스트 시나리오 추출 알고리즘

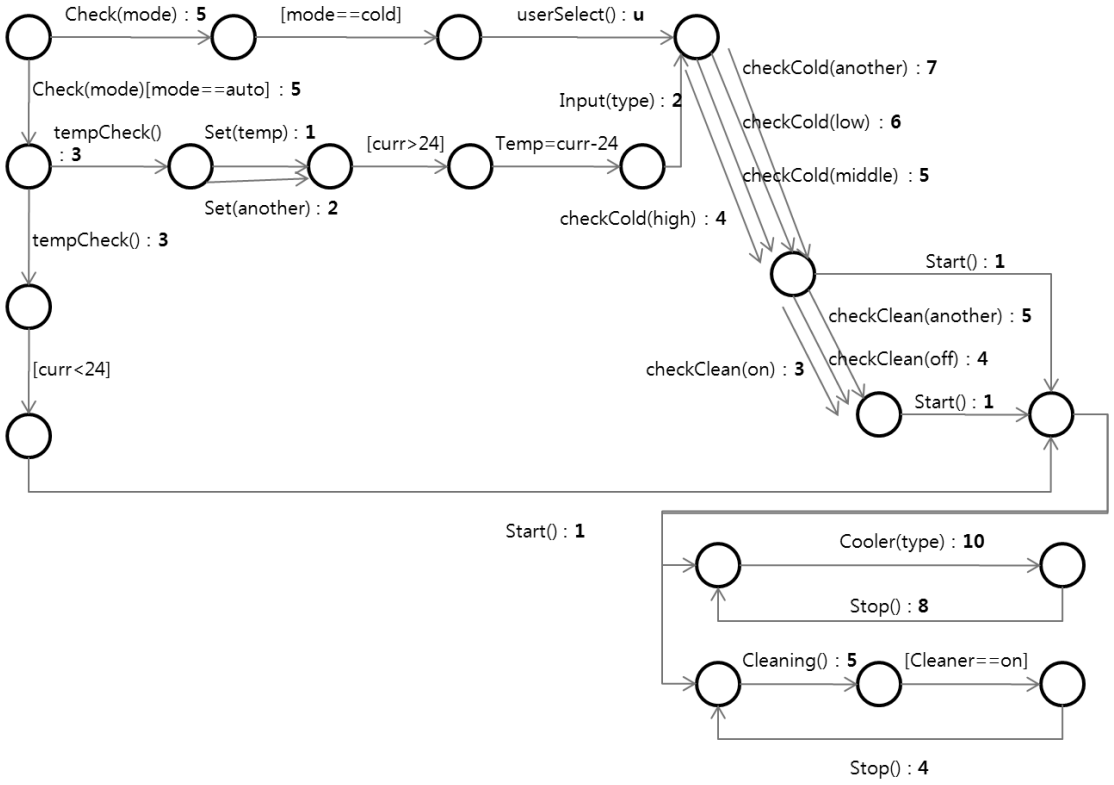
생성된 테스트 모델은 간단한 가중치 그래프 형태로 표현되어 있다. 본 연구에서 추출하고자 하는 최적화된 테스트 시나리오는 테스트 모델에서 가중치를 고려하여 최단 거리를 가지는 경로를 의미한다. 즉, 초기 상태인 한 정점에서 모든 테스트 케이스를 지나는 최단 거리의 경로들을 찾는 것이다. 한 정점에서 모든 정점으로 가는 최단 거리를 찾는 방법인 Dijkstra 알고리즘[12]을 변경하여 테스트 시나리오를 추출하는 알고리즘으로 이용한다. 테스트 시나리오를 추출하기 위해서 추가로 고려해야 할 사항은 다음과 같다.

- 테스트 케이스의 간선이면 가중치에 상관 없이 경로에 추가한다.
- 여러 테스트 케이스 간선이면 최단 경로를 간선 수만큼 복제한다.
- 간선의 가중치가 사용자 입력 가중치일 때 테스트 케이스 간선이 아니면 무조건 회피한다.

알고리즘 1은 Dijkstra 알고리즘을 변경하여 최단 거리의 테스트 시나리오를 추출하는 알고리즘을 나타낸다.

4. 기존의 시나리오 추출 방법과의 비교

본 장에서는 기존의 적합성 테스트 시나리오 추출 방법인 T-method, DS-method, UIO-method와 이 연구에서 제안한 수행시간 및 사용자 입력을 고려한 테스트 시나리오 추출 방법을 비교한다. 기존의 FSM 모델을 이용한 테스트 시나리오 추출 방법들은 각 함수들의 수행시간을 고려하지 않고



(그림 10) 생성된 테스트 모델

[알고리즘 1] 최적 테스트 시나리오 추출 알고리즘

설명 : 가중치 포함 방향 그래프에서 모든 테스트 케이스를 지나는 최단 경로 집합을 구하라.
입력 : 정점이 n(>=2)개 있는 연결된 가중치 방향 그래프, 그래프는 2차원 배열weight[][], tMatrix[][]로 표현되며, 행과 열의 인덱스는 각 각 0 부터 n-1까지이다. 여기서 [i][j]는 i번째 정점에서 j번째 정점을 잇는 이음선상의 가중치가 된다.
출력 : 모든 테스트 케이스를 지나는 최단 경로 집합

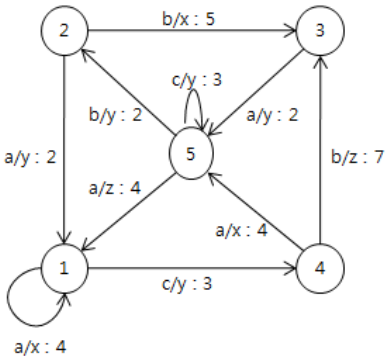
```
void testScenarioExtracion(int n, Node initNode) {
    Edge e;
    ArrayList <Edge> F;
    // 0번 인덱스는 시작 정점이기 때문에 최단 경로에 대한 배열은 1부터 시작한다.
    int touch[1 .. n-1];
    int length[1 .. n-1];

     for(int i = 1; i < n; i++) {
          touch[i] = 0;
          length[i] = weight[0][i];
     }
     for(int j = 1; j < n; j++) {
          int min = length[1];
          for(int i = 2; i < n; i++) {
               if(length[i] >= 0 && length[i] < min) {
                    min = length[i];
                    vnear = i;
               }
          }
          // 모든 정점으로 가는 최단 경로 간선 집합
          e = tMatrix[touch[vnear]][vnear];
          F.add(e);
          for(int i = 1; i < n; i++) {
               if(tMatrix[vnear] is test case) {
                         // 간선이 테스트 케이스일 경우
                    if(test cases) {
                              // 테스트 케이스가 여러 개이면
                              for( int j = 0; j < num of test cases; j++) {
                                        // 테스트 케이스 수 만큼 시작 정점에서 오는
                                   // 최단 경로를 복사한다.
                                        Copy(path);
                              }
                         }
                         length[i] = length[vnear] + weight[vnear][i];
                         // 중간 단계에 속하지 않는 각 정점에 대해서 최단 경로를
                   // 바꾼다.
                          touch[i] = vnear;
               }
                    else if((length[vnear]+weight[vnear][i]) < length[i]) {
                         length[i] = length[vnear] + weight[vnear][i];
                         touch[i] = vnear;
                    }
                    length[vnear] = -1;
          }
     }
}
```

있다. 또한 알고리즘을 적용하는 모델이 외부에 보여지는 기능들만을 표현한 프로토콜 상태 머신 형태로 되어있다. 이 연구에서 제안하는 알고리즘은 외부에 보여지는 API 함수의 수행시간뿐만 아니라 API 함수를 테스트 하기 위해 내부의 상태를 진행하는 부분에서도 수행시간에 대한 가중치를 고려하여 실제 테스트 시간을 줄이고자 하였다.

(그림 11)은 Kim의 연구[13]에서 기존 적합성 테스트 시나리오 추출 방법들을 비교하기 위한 유한 상태 머신이다. 각 간선들은 테스트 대상이 되는 테스트 케이스들을 의미한다. 그러므로 테스트 시나리오를 추출할 때 모든 간선을 포함하도록 해야 한다. 알고리즘을 적용하기 위해 각 테스트 케이스마다 가중치를 부여하였다.

또한 내부 상태들에 대해서도 최단 경로를 찾아야 하기 때문에 (그림 11)의 정점 2, 3, 4에 (그림 12)와 같은 내부 상태를 정의하였다.

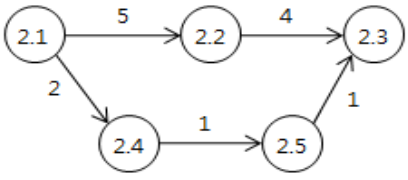
본 연구에서 제안하는 알고리즘은 내부 상태에서 최단 경



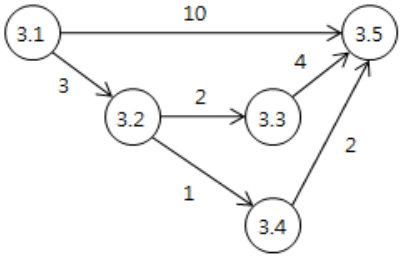
(그림 11) 알고리즘 수행시간 비교를 위한 비교 모델

로를 찾아가기 때문에 (그림 12)에서 보이는 내부 상태 중에서 수행시간 가중치가 가장 작고, 사용자 입력 부분을 피하는 경로로 테스트 시나리오를 추출한다. 하지만 기존의 방법들은 내부 상태에 대해 고려하지 않기 때문에 최악의 경우, 수행시간이 가장 높고, 사용자 입력을 지나가는 경로로 테스트를 수행할 수도 있다. <표 2>는 기존의 방법들과 제안하는 알고리즘을 비교한 것이다.

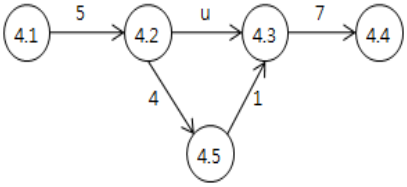
기존의 테스트 시나리오 추출 알고리즘은 내부 상태에서의 진행을 고려하지 않기 때문에 실제 테스트를 수행할 때 테스트 대상 함수까지의 내부 상태 진행은 최악의 경우 수행시간 및 사용자 입력 가중치가 가장 큰 경로로 진행될 수 있다. <표 2>와 같이 기존의 T-method, DS-method,



(a) 정점 2의 내부 상태



(b) 정점 3의 내부 상태



(c) 정점 4의 내부 상태

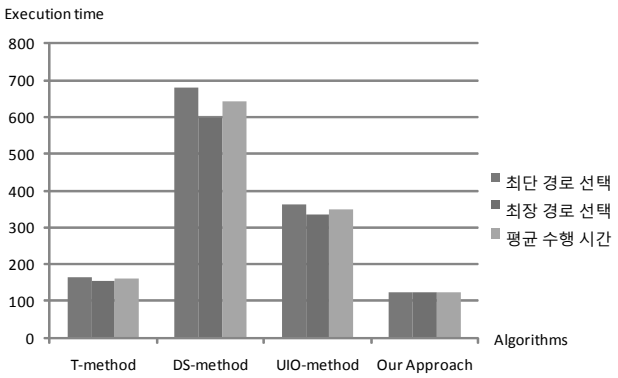
(그림 12) 비교 모델의 내부 상태

〈표 2〉 제안하는 알고리즘과 기존 알고리즘의 비교 표

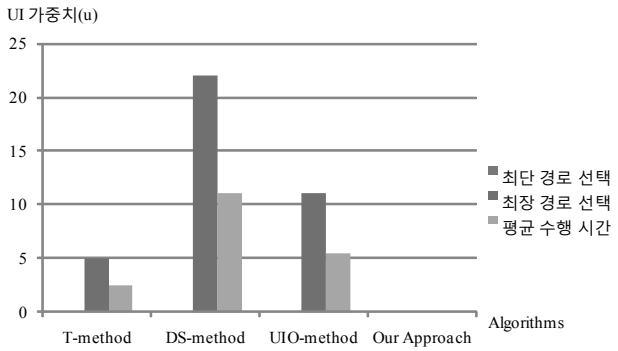
테스트 시나리오 추출 방법	최단 경로 선택(총 가중치)	최장 경로 선택(총 가중치)	평균 가중치
T-method	166	155 + 5u	159.9 + 2.5u
DS-method	682	599 + 22u	641.3+11u
UIO-method	364	336 + 11u	350.5+5.5u
Our Approach	124	124	124

UIO-method에서 테스트 시나리오를 추출할 때 최단 경로를 선택할 경우와 최장 경로를 선택할 때 테스트 수행시간은 차이를 보인다. 하지만 제안한 테스트 시나리오 추출 알고리즘을 이용하면 내부 상태에서 가중치가 최소가 되는 경로를 찾아 테스트 시나리오를 추출하기 때문에 어떤 상황에서도 수행시간 가중치가 최소가 되는 경로를 찾아 테스트 시나리오를 추출한다. 또한 기존의 방법들이 내부 상태에서 최단 경로로 진행하였다고 가정하더라도 제안하는 테스트 시나리오 추출 알고리즘은 불필요한 테스트 케이스의 반복을 줄이고 최소한 한 번 이상 테스트 대상 함수를 수행하는 최단 경로를 추출하기 때문에 실제 테스트 수행시간을 훨씬 줄일 수 있다. 그리고 테스트 대상이 아닌 사용자 입력을 피함으로써 테스트 수행시간을 줄이고 테스트 수행 시 자동화율을 높일 수 있다. (그림 13)과 (그림 14)는 테스트 시나리오 추출 기법들에 대한 비교 결과를 그래프로 나타낸 것이다.

(그림 13)에서 각 알고리즘이 내부 상태의 최장 경로에 대한 시나리오를 추출했을 때 수행시간이 최단 경로를 선택



(그림 13) 테스트 시나리오 수행시간 비교 그래프



(그림 14) 테스트 시나리오 사용자 입력 비교 그래프

했을 때보다 작게 나오지만 (그림 14)의 사용자 입력 가중치를 고려하면 실제로 테스트 수행시간은 더 많이 걸린다는 것을 알 수 있다. 사용자 입력이 존재하면 앞에서 기술한 바와 같이 테스트 수행을 자동화할 수 없고, 실제로 사람이 입력하는 시간 동안 대기해야 하기 때문에 테스트 수행시간이 길어지게 된다.

본 연구에서 제안하는 알고리즘은 기본적으로 최단거리 경로를 선택하기 때문에 (그림 14)에서 나타나는 것처럼 불필요한 사용자 입력을 피하여 테스트 자동화를 향상시킬 수 있다. 또한 기존의 방법들이 내부 상태에서 최단 경로를 지나가더라도 제안하는 알고리즘을 이용하면 모든 테스트 케이스를 포함하는 최단 경로를 찾기 때문에 테스트 수행시간이 작은 테스트 시나리오를 추출할 수 있다.

5. 결 론

이 연구에서는 임베디드 소프트웨어의 적합성 테스트 수행 시에 테스트 수행시간과 사용자 입력을 고려하여 최적화된 테스트 시나리오를 추출하는 기법을 제안하였다. 기존의 적합성 테스트 시나리오는 실제 테스트 수행시간에 대해 고려하지 않고, 테스트 자동화를 저하시키는 사용자 입력에 대한 처리가 없어 테스트 수행의 효율을 떨어뜨린다. 또한 단순히 테스트 케이스를 한 번 이상 수행하는 것에만 초점을 맞추고 있어 테스트의 입력이 되는 파라미터의 종류에 따라 테스트를 수행할 수가 없다.

제안하는 테스트 시나리오 추출 기법을 이용하면 기존의 방법과는 달리 적합성 테스트 대상이 되는 외부 인터페이스와 내부 함수들의 수행시간을 고려하여 외부 인터페이스를 모두 포함하면서 수행시간이 가장 적게 걸리는 시나리오를 추출할 수 있다. 또한 테스트 자동화를 저하시키고 테스트 수행시간을 증가시키는 불필요한 사용자 입력을 피하여 테스트 시나리오를 추출함으로써 테스트 자동화를 향상시킬 수 있다. 그리고 테스트 케이스를 한 번 이상 수행하는 것에만 초점을 맞춘 기존의 방법과 달리 테스트 케이스의 입력이 되는 파라미터의 종류를 고려하여 테스트 시나리오를 추출할 수 있다.

참 고 문 헌

[1] Anders Hessel, Paul Pettersson, “A Global Algorithm for Model-Based Test Suite Generation,” *Electronic Notes in Theoretical Computer Science 190*, pp.47-59, August, 2007.

[2] 배현섭, 윤광식, 오승욱, “임베디드 소프트웨어 테스팅 이슈 및 현황,” *정보과학회지*, 제24권, 제8호, pp.40-45, 2006.

[3] OMG, Unified Modeling Language : Superstructure, version 2.1.1, 2007.

[4] Sanjai Rayadurgam, Mats P. E. Heimdahl, “Test-Sequence Generation from Formal Requirement Models,” *Proceedings of the 6th IEEE International Symposium of High Assurance Systems Engineering*, pp.23-31, 2001.

[5] R. Lai, “A survey of communication protocol testing,” *Journal of Systems and Software*, Vol.62, pp.21-46, 2002.

[6] Wang Linzhang, et al., “Generating Test Cases from UML Activity Diagram based on Gray-Box Method,” *Proceedings of APSEC'04*, pp.284-291, 2004.

[7] P. G. Sapna, H. Mohanty, “Automated Scenario Generation Based on UML Activity Diagrams,” *Proceedings of Information Technology*, pp.209-214, 2008.

[8] Philip Samuel, Anju T. Joseph, “Test Sequence Generation from UML Sequence Diagrams,” *Proceedings of the 9th ACIS International Conference on Software Engineering*, pp. 879-887, 2008.

[9] R.E. Miller, S. Paul, “Generating minimal length test sequences for conformance testing of communication protocols,” *Proceedings of Tenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol.2, pp.970-979, April 1991.

[10] Hasan Ural, Keqin Zhu, “Optimal Length Test Sequence Generation Using Distinguishing Sequences,” *IEEE/ACM Transactions on Networking*, Vol.1, pp.358-371, June 1993.

[11] ISO/IEC 13210:1999(E), IEEE Std 2003, Information Technology - Requirements and Guidelines for Test Methods Specifications and Test Method Implementations for Measuring Conformance to POSIX Standards, 1999.

[12] Richard E. Neapolitan, Kumarss Naimipour, *Foundations of Algorithms Using C++ Pseudocode 3rd Edition,* Jones and Bartlett Publishers, 2004.

[13] Chul Kim, J. S. Song, “Test Sequence Generation Methods for Protocol Conformance Testing,” *Proceedings of Computer Software and Applications Conference 1994*, pp. 169-174, November, 1994.

박 인 수

e-mail : ispark82@gmail.com

2008년 경북대학교 전자전기컴퓨터학부 졸업 (학사)

2010년 경북대학교 전자전기컴퓨터학부 졸업 (공학석사)

2010년~현 재 삼성전자 무선사업부 사원

관심분야 : 임베디드 소프트웨어 모델링, 소프트웨어 테스팅, 차량용 운영체제

신 영 술

e-mail : youngsulshin@gmail.com
2005년 경북대학교 전자전기컴퓨터학부 졸업 (학사)
2007년 경북대학교 컴퓨터과학과(이학석사)
2007년～현 재 경북대학교 전자전기컴퓨터학부 박사과정
관심분야 : 임베디드 소프트웨어 모델링 및 분석, 소프트웨어 테스팅, 시뮬레이션

안 성 호

e-mail : ahnsh@etri.re.kr
1995년 2월 광운대학교 전자공학과(학사)
1999년 8월 광운대학교 전자공학과(석사)
2000년～현 재 한국전자통신연구원(ETRI) 대경권연구센터 자동차SW플랫폼연구팀 선임연구원
관심분야 : 임베디드 SW 분야, 자동차 전장 분야, SW 테스팅 분야

김 진 삼

e-mail : jinsam@etri.re.kr
1984년 2월 중앙대학교 전자계산학과(학사)
1986년 8월 중앙대학교 전자계산학과(석사)
2006년 2월 대전대학교 컴퓨터공학과(박사)
1987년～현 재 한국전자통신연구원(ETRI) 대경권연구센터 자동차SW플랫폼연구팀 책임연구원
관심분야 : SW 프로세스, SW 품질

김 재 영

e-mail : jaeyoung@etri.re.kr
1991년 2월 경북대학교 컴퓨터공학과(학사)
1993년 2월 경북대학교 컴퓨터공학과(석사)
1993년～1999년 LG정보통신(주)
2000년～현 재 한국전자통신연구원(ETRI) 대경권연구센터 자동차SW플랫폼연구팀 팀장
관심분야 : 임베디드 SW 분야, 자동차 전장 분야

이 우 진

e-mail : woojin@knu.ac.kr
1992년 경북대학교 컴퓨터과학과(학사)
1994년 한국과학기술원 전산학과(공학석사)
1999년 한국과학기술원 전산학과(공학박사)
1999년～2002년 한국전자통신연구원 S/W 공학연구부 선임연구원
2002년～현 재 경북대학교 전자전기컴퓨터학부 부교수
관심분야 : 임베디드 시스템 모델링 및 분석, Formal methods, CBD, Requirements Engineering, Petri nets 등