MSCTest: 내장 소프트웨어 테스트를 위한 자동화 도구

(MSCTest: An Automated Testing Tool for Embedded Software)

이 남 희 † 서 선 애 † 김 태 효 † 차 성 덕 †† 이 재 원 ††† 박 기 웅 †††

(Nam Hee Lee) (Sun Ae Seo) (Tae Hyo Kim) (Sung Deok Cha) (Jae Won Lee) (Ki Woong Park)

요 약 내장 소프트웨어는 외부 입력과 시스템의 현재 상태를 함께 고려하여 출력을 결정하는 특성을 가지고 있기 때문에, 하나의 모듈을 테스트하기 위해서는 해당 모듈에 대한 단위 테스트 기법 이외에, 추가적으로 그 모듈에 도달하기 위한 다양한 시퀀스를 생성하는 방법이 필요하다. 본 논문에서는 내장 소프트웨어의 모듈 테스트를 위한 명세 기반의 테스트 방법을 제안하고 이를 지원하는 자동화 도구인 MSCTest를 구현하였다. 각 모듈의 기능은 결정표 (decision table)를 이용하여 명세하고, 시스템에 발생 가능한 시나리오는 데이타 표현을 첨가한 MSC (Message Sequence Charts)로 기술하여 테스트 시퀀스를 생성한다. MSCTest는 MSC 그래픽 편집기, 테스트 시퀀스와 데이타 생성기, 그리고 생성된 테스트 데이타를 수행시키는 테스트 드라이버 생성기로 구성되어 있다. MSCTest를 이용하여 웹 비디오폰이나 디지털 TV 등에서 편리한 사용자 인터페이스를 제공하기 위해 사용되는 EsWin (Embedded System Windows) 소프트웨어를 효과적으로 테스트할 수 있었다.

Abstract Embedded software generates its outputs using current states of the system as well as external inputs. When a module in embedded software is tested, we need an automated testing tool, which generates possible sequences to reach the module as well as input data of the module, to reduce the testing time and to improve the quality of software. In this paper, we use decision table to specify the functionality of the module and data-annotated MSC (Message Sequence Charts) to describe scenarios, and implement a tool, which we call MSCTest, to automate the testing process. MSCTest consists of MSC graphic editor, test sequence and data generator, and test driver generator. MSCTest is effectively applied to test EsWin which is a kind of window library used in embedded systems.

1. 서 론

내장 소프트웨어 (Embedded Software)는 시스템에 내장되는 컴퓨터 상에서 동작하는 소프트웨어로서, 시스템 하드웨어로부터 입력을 받아 적절한 제어 신호를 발생하는 기능을 수행한다. 근래에 들어서는 내장 소프트웨어에도 그래픽 사용자 인터페이스와 같이 다양하고 복잡한 기능에 관한 요구사항이 증가하고 있다. 이러한 내장 소프트웨어는 외부의 입력에 의해서만 출력을 결정하는 것이 아니라 시스템의 현재 상태도 같이 고려하여 출력을 결정하는 특성이 있다. 즉, 내장 소프트웨어에서 한 모듈의 수행 결과는 그 모듈의 입력 파라미터에 의해서만 결정되는 것이 아니라, 그 모듈이 호출되기까지의 경로에 많은 영향을 받는다. 하지만, 기존의 테스트 자동화 도구들에서는 이러한 특성을 잘 고려하고 있지 못하고 있다.

지금까지 개발된 테스트 자동화 도구는 테스트 관리를 도와주는 도구에서부터 테스트 데이타를 생성하는 도구까지 다양하다[1]. 하지만, 대부분의 도구들은 Windows나 Unix 기반의 소프트웨어에 관한 테스트를 수행하기 때문에, 내장 소프트웨어의 테스트 환경을 지

† 비 회 원 : 한국과학기술원 전산학과
nhlee@salmosa.kaist.ac.kr
saseo@salmosa.kaist.ac.kr
taihyo@salmosa.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
cha@salmosa.kaist.ac.kr
††† 비 회 원 : (주)삼성전자 중앙연구소 S/W센터 연구원
jwlee@swc.scc.samsung.co.kr
kwpark@swc.scc.samsung.co.kr
논문접수 : 1999년 6월 28일
심사완료 : 2000년 2월 10일

원하지 못한다. 대표적인 내장 소프트웨어용 테스트 자동화 도구로는 Prosoft의 SwifTest[2]나 ATTOL Testware의 UniTest[3]등이 있다. SwifTest는 Statecharts로 기술된 시스템 모델로부터 자동으로 테스트 시퀀스를 생성하고 수행하지만, 각 모듈의 기능에 관해서는 고려하지 못한다. 또한, SwifTest에서는 Statecharts를 사용하기 때문에 시스템의 전체 행위에 대하여 기술한 모델이 필요하다. UniTest는 SwifTest와는 달리 코드를 기반으로 하는 테스트 도구로서, 각 모듈의 코드 분석을 통하여 테스트 데이타를 생성하고 수행하지만, 그 모듈까지의 도달 경로, 즉 테스트 시퀀스를 생성하지는 못한다.

테스트 시퀀스를 생성하는 기존 연구로는 Mandrioli [4], Caver와 Tai[5], 그리고 Bae[6]의 연구 등이 있었다. Mandrioli 등은 TRIO 로직으로 기술된 실시간 시스템 명세의 만족성 (satisfiability)을 검사하기 위해 각 TRIO 식을 해석하는 과정에서 발생하는 이벤트 히스토리를 시스템의 테스트 시퀀스로 사용하였고[4], Carver와 Tai는 병행 프로그램에 무수히 많이 발생할 수 있는 동기화 시퀀스를 프로그램의 실행을 통해서 구하고, 그 중에서 필요한 것들만 추출하기 위해 CSPE (Constraints on Succeeding and Preceding Events)를 이용하여 제한하거나 추가로 생성하는 방법을 제안하였다[5]. Bae는 MSC (Message Sequence Charts)로 병렬 프로그램의 메시지 전송 관계를 명세하고, 이로부터 발생 가능한 메시지 전송 시퀀스를 생성하는 방법을 제안하였다[6]. 하지만, 세 방법 모두 각 모듈의 기능에 관해서는 고려하지 않고 있다.

본 논문에서는 내장 소프트웨어 테스트를 자동화할 수 있는 도구인 MSCTest를 제안하고 구현하였다. MSCTest는 각 모듈의 기능 테스트를 함께 고려한 테스트 시퀀스 자동생성은 물론, 테스트 실행 코드 생성, 그리고 테스트 수행 결과 분석을 자동으로 수행한다. MSCTest의 입력으로는 결정표 (decision table)와 MSC가 사용된다. 결정표를 이용하여 각 모듈의 단위 테스트를 위한 기능을 기술하고, MSC를 이용하여 각 모듈의 도달 경로를 시나리오 형식으로 기술한다. MSCTest는 기술된 명세를 입력으로 받아 자동으로 각 모듈의 파라미터 값과 그에 따른 결과 값에 관한 오라클을 포함하는 테스트 스크립트를 생성하고, 또한 각 모듈의 시그너처를 이용하여 테스트 스크립트를 자동 수행할 수 있는 드라이버 코드를 생성한다. 이렇게 생성된 코드는 내장 소프트웨어와 함께 컴파일 되어 테스트를 수행할 수 있는 태스크의 형태를 가지며, 실행된 결과의 옳고 그름은 자동으로 판별된다.

MSC는 통신 소프트웨어 개발 분야에서 주로 사용되어 왔기 때문에, 이를 기반으로 하는 테스트 방법들은 주로 통신 소프트웨어 개발 도구의 일부로 포함되어 있다. 대표적인 도구로 Telelogic의 Tau[7]와 Verilog의 ObjectGEODE[8]가 있다. 두 도구 모두 MSC를 기반으로 테스트 시퀀스를 생성하여 TTCN (Tree and Tabular Combined Notation)의 형태로 저장되고, 이때 필요한 데이타는 ASN.1을 이용하여 기술한다. 테스트 시퀀스 생성 방법에서는 MSCTest와 유사하지만, 실제 데이타 생성을 위해 필요한 다양한 형태의 자료를 ASN.1에서는 기술할 수 없고, 또한 여러 함수들 간의 데이타 의존성을 표현할 수도 없다.

MSCTest를 이용하여 웹 비디오폰이나 디지털 TV 등에서 편리한 사용자 인터페이스를 제공하기 위해 사용되는 EsWin (Embedded System Windows)을 테스트하였다. EsWin의 총 200여 개의 함수들 중에서 대표가 되는 함수 50개를 선정하여 MSC와 결정표로 명세하고 테스트를 수행하였다. 테스트 수행 결과 기존의 수작업 테스트에 비해 더 많은 오류를 찾는다거나, 시간을 많이 단축할 수는 없었다. 그 이유는 대상으로 사용한 시스템이 이미 여러 번의 수작업 테스트를 수행한 상태였고, 초기 명세를 준비하는데 많은 시간이 필요하기 때문이다. 하지만, 버전 변경에 따른 회기 (regression) 테스트에서는 수작업의 경우보다 나은 결과를 나타낼 것으로 기대된다.

본 논문은 다음과 같이 구성된다. 2장에서는 MSC를 이용한 테스트 시퀀스 생성과 결정표를 이용한 테스트 데이타 선택 방법에 대해서 기술한다. 3장에서는 테스트 대상 시스템인 EsWin에 대해 간략히 소개하고, MSCTest 구현과 적용 결과에 대해서 기술하고, 마지막으로 4장에서 결론 및 향후 연구 방향에 대해서 기술한다.

2. 접근 방법

2.1 MSC를 이용한 테스트 시퀀스 생성

MSC는 수직선과 화살표와 같은 간단한 기호들을 이용하여 시스템을 구성하는 콤포넌트들 사이의 상호 작용을 표현하고 있기 때문에, 직관적으로 이해하기 쉽고 작성하는 것이 간단하다. MSC는 통신 소프트웨어 분야는 물론 다양한 시스템에 적용되었고, 또한 소프트웨어 공학의 다양한 분야 - 시스템의 요구사항 기술 [9], 테스트 시나리오 기술 [6], SDL 명세 검증을 위한 시스템의 성질 기술 [10] 등 - 에서 사용되고 있다.

ITU-T에서 MSC'96 표준안을 정의한 Z.120 [11]은 시스템을 구성하는 각각의 컴포넌트를 인스턴스 (또는 프로세스)로 나타내고, 그들 간의 상호 작용을 메시지로 표현하고 있는 Basic MSC (bMSC)와 bMSC 사이의 관계를 계층적으로 표현하고 있는 High-Level MSC (hMSC)로 구성된다. hMSC를 정의함으로써 부분적인 행위만을 기술하는 기존 MSC의 단점을 보완할 수 있다. MSC'96은 hMSC 이외에도 반복적인 수행을 나타내는 loop나 선택적인 수행을 나타내는 alt등 다양한 개념들을 제공하고 있다.

한 모듈의 수행시 전에 수행된 모듈의 결과가 사용되어야 하는 경우가 많이 있다. 예를 들어, 메인 윈도우를 생성한 후 그 안에 버튼과 같은 컨트롤 윈도우를 생성해야 의미있는 실행이 된다. 따라서, 이러한 파라미터 값 전달을 위해, Z.120에서는 지원되지 않는 데이타 표현 방법을 확장하여 사용한다. Z.120에는 메시지의 파라미터 변수의 표현은 지원하고 있지만, 특정 값을 주는 것을 지원하지 않는다. 본 논문에서는 값을 전달하기 위해 'parameter[variable]' 또는 'parameter['value']의 형식을 사용한다. 여기서 'parameter'는 다음 모듈에 나타나는 파라미터 시그너쳐 중의 하나이고, 'variable'은 전에 수행된 모듈의 결과 값을 저장하고 있는 변수이다. 그리고, 'value'는 특정한 값을 주기를 원할 때 사용할 수 있다.

위와 같은 방법으로 기술된 MSC로부터 테스트 시퀀스를 작성하는 과정은 [6]의 연구를 이용한다. [6]의 테스트 시퀀스 생성 방법을 간단히 요약하면 다음과 같다. 우선 테스트를 위하여 MSC 내의 프로세스들을 두 개의 그룹으로 나눈다. 한 그룹은 테스트 대상 프로세스 (IUT - Implementation Under Test)이고, 또 다른 그룹은 IUT를 구동하기 위한 프로세스 (TM - Test Manager) 이다. 그리고, 각 그룹 내의 이벤트들에 대해서 논리 시간표를 부여한다. TM에서 IUT로의 메시지들은 순차적 혹은 병행적으로 전송될 수 있다. 이때, 각 프로세스 내부의 이벤트들 간에는 순서 관계가 존재하며, 메시지를 송, 수신하는 프로세스 사이에도 이벤트의 순서 관계가 있게 된다. 이러한 순서 관계를 정하는 방법으로 논리 시간표를 이용한다. 논리 시간표 부여 방법에 관한 자세한 내용은 [6]을 참조하기 바란다. 마지막으로 각 이벤트에 부여된 논리 시간표에 의한 순서 관계를 위반하지 않도록 하면서 기술된 MSC에서 발생 가능한 이벤트 시퀀스를 생성하여 이를 테스트 시퀀스로 한다. 한 MSC내에는 병행하여 발생할 수 있는 이벤트들이 존재하기 때문에, 하나의 MSC에서 여러 개의 시퀀스가 생성될 수 있다.

2.2 결정표를 이용한 테스트 데이타 생성

결정표는 상업 데이타 처리 분야에서 널리 사용되고 있는 명세 방법으로[12], 의미를 이해하기 쉽고 일치성 (consistency)과 완전성 (completeness) 판별이 쉽다. 표 1에서 보는 것과 같이 결정표의 행은 조건 (condition)과 행위 (action)를 나타내는 두 부분으로 구성되고, 열은 각 조건의 조합으로 가능한 규칙 (rule)으로 구성된다. 또한, 각 규칙의 조건에 해당되는 칸이 'T'이면 조건이 만족함을 'F'이면 만족하지 않음을 의미한다. 그리고, 'X'는 그 조건이 만족하든지 않든지 상관없음을 의미한다. 행위 부분은 규칙에서 지정한 조건이 만족되면 발생해야 하는 것들을 표현하는 것으로, 역시 'T'와 'F'로 표시한다.

표 1 결정표 예제 : 모듈 GetShowFlag()

	Para #	Range	R1	R2	R3	R4	R5
condition	WID wid	From 0 To MaxWindowNum-1	T	T	T	T	F
	Short WindowInfo[wid].showState	ST_INIT(0)	T	F	F	F	X
		ST_SHOW(1)	F	T	F	F	X
		ST_ICON(2)	F	F	T	T	X
		ST_HIDE(3)	F	F	F	F	X
action	return	ST_INIT	T	F	F	F	F
		ST_SHOW	F	T	F	F	F
		ST_ICON	F	F	T	F	F
		ST_HIDE	F	F	F	T	F
		Don't care	F	F	F	F	T

각 모듈에 관한 기능을 결정표를 이용하여 명세하고 이로부터 테스트 데이타를 생성하는 과정은 다음과 같다.

- 전체 입력 변수와 전체 출력 변수를 찾아낸다.

모듈의 수행에 영향을 미치는 입력 변수로는 모듈의 입력 파라미터와 전역 변수가 있고, 모듈에 의해 영향을 받는 출력 변수로는 출력 파라미터와 전역 변수 그리고 리턴 값이 있다. 전역 변수에 관한 정보는 테스터가 자연어 명세로부터 얻을 수 없기 때문에, 전역 변수에 관한 것도 고려하려면 개발자의 도움을 받아야 한다.

- 전체 입력 변수를 이용하여 조건을 찾아낸다.

모듈의 수행에 영향을 미치는 전체 입력 변수들이 결정되면, 각 변수 별로 가능한 값의 범위를 나눈다. 각각의 범위는 하나의 조건이 된다. 각 조건들은 다른 변수

의 조건과 함께 출력 값에 영향을 미친다. 따라서 가능한 출력의 종류는 전체 입력 변수로부터 가능한 조건들의 조합으로 나타날 수 있으나, 실제로 영향을 미치는 조건의 조합은 그보다 적다.

• 전체 입력 변수들의 조건에 따라 모듈의 명세에 맞게 'T', 'F' 또는 'X' 값을 적용하여 결정표의 규칙을 완성한다.

전체 입력 변수가 m개일 때, 입력 변수 v_i ($1 \le i \le m$)가 $r_{i,j}$ ($1 \le j \le n$, n은 변수 v_i가 가지의 범위의 개수)의 범위를 갖고, 전체 입력 변수들의 범위의 개수가 동일하다고 가정하면, $m \times n$개의 규칙이 필요하다. 하지만, 모든 규칙에 대해 서로 다른 출력 값이 결정되어질 필요가 없기 때문에, $m \times n$보다 적은 수의 규칙만으로 결정표를 구성할 수 있다. 그리고, 각 규칙들에 대해서 $r_{i,j}$는 다음을 만족한다.

$$\bigvee_{1 \le j \le n} r_{i,j} = true$$

하지만, 임의의 p, q에 대해서 $r_{i,p} \wedge r_{i,q} = false$일 필요는 없다. 한 입력 변수 내의 여러 개의 범위가 $true$인 것은 서로 다른 규칙으로 기술하여야 하지만, 하나로 사용하여도 그 의미가 변하지 않기 때문이다.

• 각 규칙 별로 출력 변수들의 가능한 값을 결정해 준다.

결정표를 만드는 마지막 단계는 전 단계를 통해 생성된 조건 테이블의 각 규칙에 대해서 전체 출력 변수 값을 정해주는 과정이다. 즉, 주어진 전체 입력 변수 값과 모듈에 의해 수행이 마쳐진 출력 결과에 대한 올바른 범위를 여기서 표시하게 된다.

• 작성된 결정표에서 테스트 데이타를 생성한다.

결정표의 하나의 규칙을 만족하는 각 입력 변수들의 값이 하나의 테스트 데이타를 구성한다. 이때, 범위의 형식을 가지는 변수 (표 1의 wid)의 경우에는 기존의 영역 테스트 방법을 이용하여 테스트 데이타를 구한다.

표 1은 GetShowFlag()라는 함수 (모듈)를 결정표를 이용하여 표현한 것이다. GetShowFlag() 함수는 윈도우 id를 받아서 해당하는 윈도우의 show 상태를 되돌려 주는 함수로, *Short GetShowFlag(WID wid)*와 같이 사용된다. 표 1에서 이 함수의 전체 입력 변수에 wid이외에 WindowsInfo[]라는 전역 변수가 사용되고 있는데, 이 변수는 윈도우에 대한 정보를 가지고 있는 구조체로 시그너처 (signature)에는 나타나지 않지만, WindowsInfo[i].showState에 i번째 윈도우의 show 상태가 저장되기 때문에 이 함수에서 참조하는 전역 변수이다. 표 1은 wid가 0에서 MaxWindowNum - 1 사이에 있는 정수이고, WindowsInfo[wid]. showState는 4가지 값을 가질 수 있음을 표현하고 있다. 또한, 함수의 전체 출력 변수는 리턴 값 하나만 존재하는데, 리턴 값은 WindowsInfo[wid].showState의 값에 의해 결정된다.

2.3 테스트 스크립트 생성

MSC와 결정표를 이용한 테스트 데이타 생성의 마지막 과정은 이미 생성된 테스트 시퀀스와 결정표의 정보를 합치는 것이다. 본 연구에서는 테스트를 위해 TM에서 IUT로의 메시지 전송을 테스트하고자 하는 모듈의 수행으로, 이를 받은 IUT에서는 해당하는 모듈의 명세를 행위 (action) 문에 결정표 형식으로 표현하여 테스트 하고자 하는 모듈에 대한 데이타를 생성하게 하였다. 그리고, IUT에서 TM으로의 메시지 전송은 모듈의 실행 결과가 반환되는 것을 나타낸다. 실제로 테스트 케이스는 테스트 시퀀스에 모듈이 나타나고, 모듈을 호출하는 부분에서 결정표를 참조하여 필요한 값을 넘기게 된다. 이때 여러 개의 모듈이 연속해서 호출될 때 각 모듈에 대한 테스트 데이타들을 조합하여 스크립트의 형식으로 출력한다.

3. 도구 구현 및 적용 결과

3.1 EsWin

EsWin은 멀티미디어 단말기용 윈도우 시스템으로, 실시간 운영체제인 pSOS에서 윈도우 환경을 제공해주기 위해 시스템 크기 및 수행 시간을 최적화한 것으로, 마이크로 소프트의 Win CE와 비슷한 개념을 제공한다[13]. EsWin은 단말기용 응용프로그램 개발자들이 GUI 응용 프로그램을 개발할 수 있도록 API 함수들을 제공한다. EsWin 시스템은 기본적으로 동작하는 윈도우 매니저 태스크와 타이머 태스크, 그리고 응용 프로그램에서 제공하는 API 함수들로 구성된다. 응용 프로그램 태스크는 API 함수를 통해 EsWin 시스템을 사용하게 된다. 윈도우 매니저 태스크는 EsWin이 동작하는 동안 항상 살아있는 상태를 유지하면서, 시스템 입력 장치의 입력을 받아 처리한 후 이를 각각의 응용 프로그램 태스크에게 보내는 일을 수행한다. API 함수는 윈도우 생성과 소멸 등을 담당하는 Windows library, 그래픽 처리를 담당하는 Graphics library, 응용 프로그램을 생성과 등록, 이벤트 처리를 담당하는 Event/Task library로 구성되는데, 실제로 화면을 구성하는 것은 Optic library를 이용한다. 그림 1은 EsWin library들 사이의 계층 구조를 보여준다.

EsWin의 각 API 함수를 테스트하기 위해서는 API

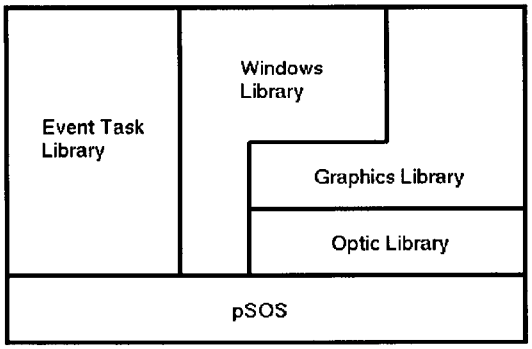

그림 1 EsWin 라이브러리의 계층 구조

함수를 호출하는 응용 프로그램 태스크가 필요하고, 이를 관리하기 위해서 윈도우 매니저 태스크에 등록하고 이벤트를 처리하는 일련의 과정들이 필요하기 때문에, Windows나 Unix 기반에서 동작하는 상용화된 자동화 도구들을 사용할 수 없다. 따라서, 수작업으로 테스트를 수행하였다. 테스터는 자연어로 기술된 라이브러리 함수에 관한 문서를 분석하고, 그들의 영역 지식을 기반으로 테스트 데이타를 선택하여 스크립트 형식으로 기술한다. 이때, 스크립트는 호출되는 함수의 시퀀스와 그때 사용되는 파라미터의 값을 포함하고 있다. 그리고, 기술된 스크립트를 수행할 수 있는 테스트 드라이버와 응용 프로그램 태스크를 작성하여 EsWin 시뮬레이터 상에서 테스트를 수행한다. 이와 같은 방법으로 두 명의 테스터가 하나의 새로운 EsWin 버전을 테스트하는데 약 한 달의 시간이 소요되고, 테스트할 때마다 계속 같은 양의 시간이 필요하였다. 또한, 테스터의 직관과 경험에 의해 테스트 데이타를 선정하기 때문에 꼭 테스트해야 할 데이타를 포함하지 못할 수 있다.

EsWin은 독립적인 함수들로 구성되어 있기 때문에, 테스트를 위해서 MSC는 윈도우 매니저의 라이브러리 함수를 호출하고 실행하고, 그 결과 값을 돌려 받아서 다른 함수의 호출에 사용하는 것을 표현할 수 있으면 된다. 따라서, 본 논문에서는 EsWin에서 생성되는 각 태스크 - 윈도우 매니저와 테스트 매니저 등 - 를 MSC의 프로세스로 나타내고, 함수 호출은 메시지, 함수의 수행은 행위 (action)를 이용하여 나타낸다.

3.2 MSCTest의 구현

앞에서 기술한 테스트 데이타 생성과정을 구현한 자동화 도구인 MSCTest는 MSC 그래픽 편집기와 테이블 편집기, 테스트 케이스 생성기로 구성되어 있고, JAVA를 이용하여 구현하였다. 그림 2는 MSCTest를 이용한 전체적인 테스트 과정을 표현한 것이다. 테스트 케이스 생성기는 MSC 그래픽 편집기를 이용하여 작성된 MSC 명세와 테이블 편집기를 이용하여 작성한 결정표를 입력으로 받아 테스트 스크립트를 자동 생성하게 된다. 이때, 생성된 테스트 스크립트를 수행하여 그 결과를 비교하고 저장할 수 있는 C 언어 형식의 테스트 드라이버 코드도 생성한다. 생성된 테스트 드라이버 코드는 EsWin 개발 팀에서 제공하는 시뮬레이터와 함께 컴파일 되어 수행된다.

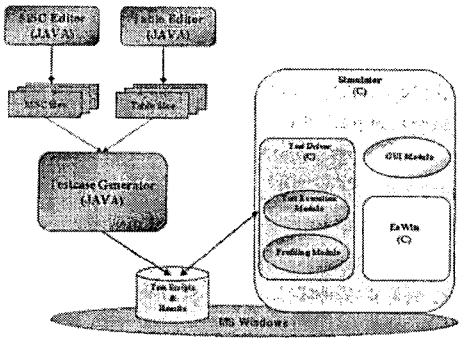

그림 2 MSCTest의 전체 구조

MSC 편집기는 테스터가 MSC를 보다 편리하게 작성할 수 있도록 도와준다. 현재 개발된 MSC 편집기는 Z.120의 bMSC에 정의된 기능과 loop, 그리고 다른 MSC의 참조 (reference) 기능을 지원한다. 작성된 MSC 명세는 Z.120의 표준 텍스트 형식 (MSC/PR)으로 저장되어 테스트 케이스 생성기의 입력으로 사용된다. MSC의 표준 텍스트 형식을 읽어 들이기 위해 JAVA로 파서를 구현하였는데, Z.120 표준의 모든 기능을 지원하기 위한 파서에서는 파싱 테이블이 너무 커져 속도가 느리다는 단점이 있다. 따라서, MSCTest에서는 bMSC만을 파싱할 수 있도록 하고, loop나 참조 기능 등은 MSC 편집기의 변환 과정에서 펼치기 (unfolding)를 수행하여 bMSC 형식으로 저장하도록 하였다.

그림 3의 MSC는 생성된 List 윈도우에 반복적으로 아이템을 추가하는 시나리오를 작성한 것이다. 테스트 대상이 되는 윈도우 매니저 (Window_Manager)와 테스트를 담당하는 테스트 매니저 태스크 (Test_Manager)를 각각 프로세스로 표현한다. 처음에 나타난 CreateMainWindow1로 명명된 둥근 사각형은 MSC의 참조를 나타내는 것으로, 다른 MSC에 자세한 기술이

있다는 것을 의미한다. 실제로 CreateMainWindow1에서는 CreateMainWindow() 함수를 이용하여 메인 윈도우를 생성한다. 다음에 있는 Main이라는 메시지는 CreateMainWindow()함수의 실행 결과를 'Main'이라는 변수에 저장하라는 뜻이고, 다음 메시지에서 이를 사용한다. 즉, 다음 메시지인 CreateControlWindow()에서 stCreate 구조체의 widParent 값으로 바로 전에 수행하여 생성한 메인 윈도우의 id 값인 'Main'을 넘겨주고 있다 (stCreate.widParent[Main]). 그리고, 파라미터에 특정 값을 넘겨주기 위해서 stCreate.Type['WINDOW_LIST']와 같이 사용하고 있다. 이것은 새로 생성되는 컨트롤 윈도우의 형태가 List임을 의미하고, 'WINDOW_LIST'는 테스트 케이스 생성기에서 결정표의 정보를 이용하여 적절한 값으로 변경하여 사용한다. 이렇게 생성된 List 윈도우의 id인 'Control'을 이용하여 AddListItem() 함수를 호출하고, GetListItemCount() 함수로 그때의 아이템 개수를 얻어낸 후, ShowWindow() 함수를 이용하여 화면에 보이는 상태를 조정한다. 이렇게 반복해서 사용될 메시지의 집합을 MSC의 loop를 이용해서 표현하고 있다. 마지막으로 DestroyWindow() 함수를 이용하여 메인 윈도우를 소멸한다.

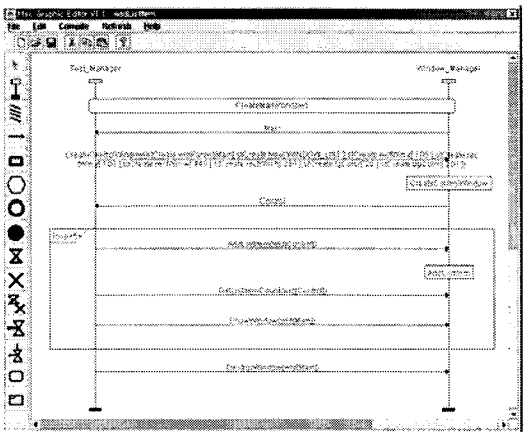

그림 3 AddListItem 시나리오

메인 윈도우나 컨트롤 윈도우를 만들기 위해서는 타이틀이나 크기, 색깔과 같은 윈도우에 대한 기본적인 정보가 필요한데, 이에 대한 명세는 결정표를 이용하여 작성한다. 그림 4는 CreateControlWindow 함수에 관한 결정표를 작성한 화면이다. 왼쪽 위에 보이는 화면에서 입출력 변수에 대한 정의를 하고, 'Table' 메뉴를 통해 결정표를 만들 면, 오른쪽 위와 같이 입력된 각 변수에 관한 영역 정보를 이용하여 결정 표 화면이 자동 생성된다. 여기에 'T', 'F', 'X' 값을 선택하여 결정 표 항목을 완성하고, 'Compile' 버튼을 누르면 아래 그림과 같은 텍스트 형식으로 표 정보가 저장되고, 이것은 테스트 케이스 생성기의 입력으로 사용된다. 현재까지 결정표에서 지원하는 데이타형은 정수형 (integer), 부울형 (boolean), 나열형 (enumeration), 실수형 (float), 스트링형 (string) 등이다.

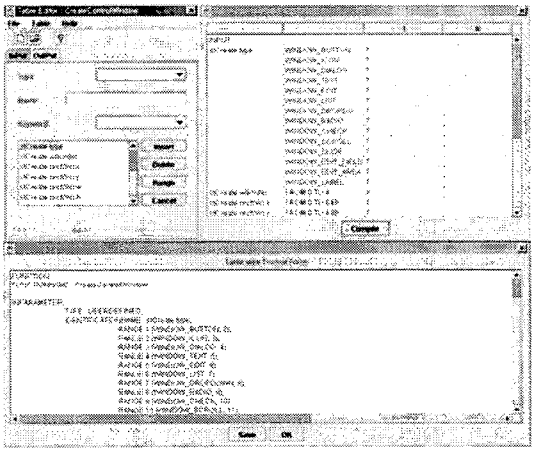

그림 4 CreateControlWindow 결정 표 명세

테스트케이스 생성기는 MSC 편집기와 테이블 편집기를 이용하여 생성된 각 명세의 텍스트 형식을 입력으로 받는다. 그리고, MSC 명세로부터 테스트 시퀀스를 생성하고, 생성된 시퀀스 상에 행위가 나타나면 해당하는 결정표를 읽어 들인다. 읽어 들이 결정표로부터 테스트 데이타를 생성하는데, 만약 그림 3과 같이 테스트 시퀀스 상에 여러 개의 결정표가 연결된다면 각각의 테스트 데이타를 조합하여 테스트 스크립트를 생성한다. 테스트 케이스 생성기에는 생성된 테스트 스크립트를 수행하고, 그 결과를 비교하고 저장하는 테스트 드라이버 생성기도 포함되어 있다. 이 과정은 테스트 케이스 생성과 독립적으로 진행되며, 각 함수의 시그너처를 입력으로 받는다.

그림 5는 테스트 케이스 생성기 화면으로 앞에서 작성한 MSC 명세를 읽어 들여 테스트 케이스를 생성하는 것을 보여주고 있다. 먼저, 'Open' 메뉴를 사용하여 MSC 명세를 열고, 'Run' 메뉴의 'Setting'을 이용하여 사용할 테스트 데이타 선택 기준을 정한다. 현재 MSCTest에서 제공하는 옵션은 각 변수의 범위 안의 값 하나 만을 임의로 선택하는 방법, 범위의 경계 값을

선택하는 방법, 앞의 두 가지 모두 선택하는 방법, 그리고 범위 안의 데이타를 20%, 50%, 100% 만큼 선택하는 6가지 방법을 제공하고 있다. 이때 모든 옵션에 대해서 범위를 벗어나는 오류 값도 포함되어 생성된다. 'Parsing'은 MSC 명세와 결정 표 명세를 파싱하여 내부적으로 필요한 자료구조를 만드는데, 이때 각 명세가 문법에 맞게 작성되어 있는 지에 관한 검사와 일치성(consistency) 검사를 수행한다. 그리고 나서 'Generate' 메뉴의 'Testcase'를 선택하면 자동으로 테스트 스크립트를 생성한다.

그림 5 테스트 케이스 생성

그림 6의 왼쪽 화면은 생성된 테스트 스크립트의 일부분을 보여주고 있다. MSC에 기술된 각 함수를 생성된 테스트 시퀀스에 따라 호출하여 하나의 테스트 과정을 완성한다. 이때, CreateMainWindow() 함수는 결정표로 그 기능이 기술되어 있으므로, 그에 해당하는 테스트 데이타를 생성하여 파라미터의 이름과 함께 저장하고, 그 결과 값이 유효하게 되는 범위를 결정표로부터 구하여 이 함수의 결과가 올바른지를 자동으로 검사할 수 있도록 한다. 각 변수 앞에 'val'로 표시된 것은 테스트 케이스 생성기에 의해 생성된 데이타이거나, MSC의 파라미터 값 지정에 의해 결정된 데이타를 의미한다. 또한, 'Main=return'과 같은 'DATASETTING'은 결과 값이 저장되어야 할 변수를 지정하는 스크립트이고, 이러한 변수들은 'var'로 지정된 변수의 데이타 처리에 사용된다. 'OUTPUTS'에 나타나는 'range 0 309'와 같은 값은, 주어진 데이타를 이용하여 성공적으로 함수가 수행되었을 때 결과 값이 가져야 할 범위를 나타내는 스크립트이다.

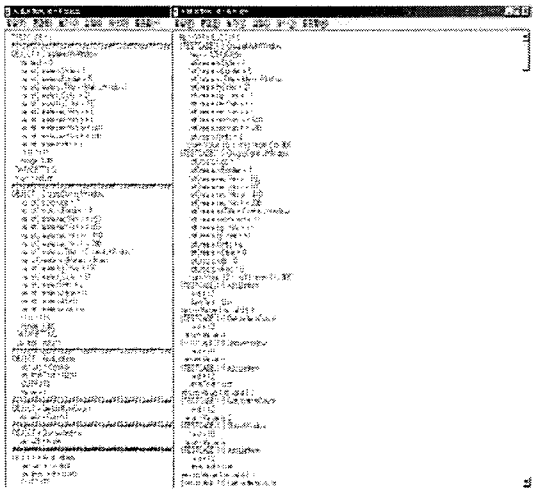

그림 6 테스트 스크립트와 결과 파일 예

테스트 케이스 생성기를 통해 생성된 테스트 스크립트는 EsWin 개발자가 제공하는 시뮬레이터를 이용하여 실제로 수행해 볼 수 있다. 이때, 테스트 케이스 생성기로부터 생성된 드라이버 코드를 함께 컴파일해 주어야 한다. 그림 7은 앞에서 작성한 MSC 명세에 따라 생성된 테스트 스크립트를 수행하는 모습으로, 메인 윈도우에 하나의 List 윈도우를 생성하고 그 안에 다섯 개의 아이템을 삽입하여 제대로 보이는 지를 테스트하고 있다. 이때, 아이템의 스트링이 처리할 수 있는 최대치 이상의 값에 대해 적절한 행동을 하는 지에 관한 테스트를 수행하기 위해 그림과 같이 dummy 스트링을 입력하고 있는 것을 볼 수 있다.

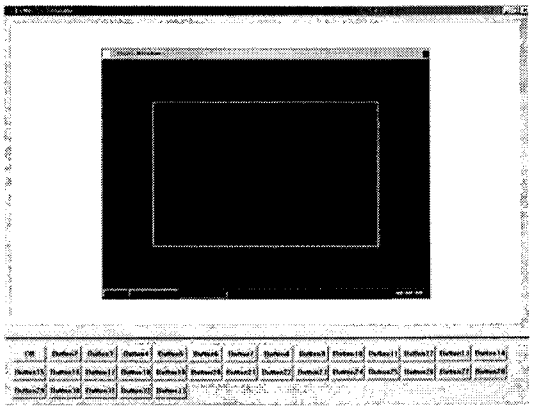

그림 7 시뮬레이터를 이용한 테스팅 실행 화면

그림 6의 오른쪽 하면은 시뮬레이터를 이용하여 그림 6의 왼쪽 테스트 스크립트를 실행한 결과이다. 실제 함수의 호출에 사용된 값을 출력하고, 그 결과 값이 원하는 범위 내에 있는지 없는지를 검사하여 'valid', 'invalid'를 출력하고 있는 것을 볼 수 있다.

3.3 적용 결과

개발된 MSCTest를 이용하여 EsWin의 총 200여 개 함수들 중에서 대표가 되는 함수 50개를 선정하여 MSC와 결정표로 명세하고 테스트를 수행하였다. 이때 선정된 함수의 대부분은 윈도우 라이브러리이다. 윈도우 라이브러리는 그 수행 결과의 옳고 그름을 자동으로 확인할 수 있지만, 그래픽 라이브러리는 사람의 눈으로 직접 확인해야 한다. 예를 들어, 원을 그리는 함수의 경우 중간에 중단된 곳은 없는 지를 자동으로 확인할 수 있는 방법은 아직 없다. 또한, 이벤트 라이브러리는 주로 마우스 등과 같은 외부 입력을 처리하는데, 매우 복잡한 가변 구조체 형식의 입력 파라미터를 사용하고 있는데, 개발된 도구에서는 기본적인 형태의 변수 - integer, boolean, character 등 - 만을 지원하고 있기 때문이다.

MSCTest를 사용해서 명세를 작성하는데 걸린 시간은 약 1주일 정도, 테스트 스크립트 생성과 실행에는 불과 몇 분밖에 들지 않는다. 하지만, 기존의 수작업 테스트와는 직접적인 비교가 불가능하다. 그것은 선정된 함수들만에 대한 수작업 테스트에서의 비교 자료가 없고, MSC나 결정표를 작성하는데 초기 비용이 많이 들기 때문이다. 즉, 새로운 방법을 도입하기 위한 교육과, 기존의 자연어로 작성된 문서를 변환하는데 드는 초기 비용이 많이 들기 때문이다. 하지만, MSCTest에서는 명세를 수정만 하면 그 후의 작업은 자동으로 진행되므로, 기존의 수작업 테스트에서 테스트 스크립트를 일일이 변경하는데 드는 시간을 절약할 수 있으므로 테스트의 효과적인 수행을 기대할 수 있다. 그리고, MSCTest를 사용함으로써 스크립트 작성에 개입할 수 있는 테스터의 오류를 막을 수 있고, 도구에서 제공하는 다양한 옵션을 이용하여 신뢰성 있는 테스트를 수행할 수 있었다.

4. 결론 및 향후 연구

내장 소프트웨어의 한 모듈을 테스트하기 위해서는 그 모듈에 대한 기능 명세는 물론, 그 모듈에 도달할 수 있는 다양한 경로에 관한 명세가 필요하다. 본 논문에서는, MSC를 이용하여 모듈의 호출 패턴을 기술하고, 결정표를 이용하여 각 단위 모듈의 기능을 기술하였다. 그리고, 기술된 명세를 기반으로 테스트 데이타를 자동 생성하는 도구인 MSCTest를 구현하고, EsWin 테스팅에 적용함으로써 도구의 효용성을 보였다. MSCTest를 EsWin의 윈도우 라이브러리 테스트에 사용함으로써 기존의 수작업으로 테스트 스크립트를 작성하고 테스트를 수행할 때보다 더 능률적으로 테스트할 수 있다.

앞으로 MSCTest를 EsWin의 그래픽 라이브러리나 이벤트 라이브러리 테스트에 효과적으로 적용하기 위해서, 가변적 구조체와 같은 좀더 다양한 자료 표현 형식과 다양한 데이타 의존성을 지원하도록 도구를 확장할 계획이다. 또한, 본 연구에서 개발된 도구는 MSC'96의 모든 기능을 제공하고 있지 못하다. MSCTest가 MSC'96을 지원한다면 좀 더 많은 시나리오를 표현할 수 있고, 도구의 범용성을 높일 수 있다. 즉, 시나리오를 합성하고, 이를 이용하여 새로운 테스트 시퀀스를 찾아냄으로써, 하나의 시나리오에 의한 테스트에서는 찾을 수 없는 좀더 다양한 시나리오를 생성할 수 있다.

참 고 문 헌

[1] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing*, pp. 417-473, Addison Wesley, 1999

[2] SwifTest, http://www.otm.fi/prosoft/

[3] ATTOL UniTest 3.3, http://www.attol-testware.com/unitest.htm

[4] D. Mandrioli, S. Morasca, and A. Morzenti, "Generating Test Cases for Real-Time Systems from Logic Specifications," *ACM Transactions on Computer Systems*, pp. 365-398, Nov. 1995

[5] R. H. Carver and K. C. Tai, "Use of Sequencing Constraints for Specification-Based Testing of Concurrent Programs," *IEEE Transactions on Software Engineering*, Vol. 24, No. 6, pp. 471-490, June 1998

[6] H. S. Bae and Y. R. Kwon, "Validation of Timing and Communication Constraints in Real-Time Parallel Programs," *Phd. Dissertation in Computer Science, KAIST*, 1999

[7] Telelogic Tau, http://www.telelogic.com

[8] ObjectGEODE, http://www.csverilog.com

[9] I. Jacobson and et al., Object Software Engineering - A Use Case Driven Approach, *Addison-Wesley*, 1992

[10] B. Algayres, Y. Lejeune, F. Hugonment, and F. Hantz, "The AVALON Projects: A Validation Environment for SDL/MSC Descriptions," *Proceedings of the 6th SDL Forum*, Oct. 1993

[11] ITU-T. Recommendation Z.120, "ITU-Telecommunication Standardization Sector," Geneva, Switzerland, May 1996

[12] B. Beizer, *Software Testing Techniques*, pp. 320-362, New York:Van Nostrand Reinhold, 1990

[13] EsWin 2.0 User's Guide, *Samsung Electronics Co.*, 1999

이 남 희
1991년 2월 한국과학기술원 전산학과 졸업(학사). 1998년 2월 한국과학기술원 전산학과 졸업(석사). 1998년 3월 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 실시간 시스템 명세 및 검증, 소프트웨어 재사용

서 선 애
1998년 2월 한국과학기술원 전산학과 졸업(학사). 2000년 2월 한국과학기술원 전산학과 졸업(석사). 2000년 3월 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 정형 명세 및 검증, 소프트웨어 테스팅, 프로그램 정적 분석

김 태 효
1998년 2월 한국과학기술원 전산학과 졸업(학사). 2000년 2월 한국과학기술원 전산학과 졸업(석사). 2000년 3월 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심분야는 정형 명세 및 검증

차 성 덕
1983년 Univeristy of California at Irvine 전산학 학사. 1986년 University of California at Irvine 전산학 석사. 1991년 University of California at Irvine 전산학 박사. 1990년 ~ 1991년 Hughes Aircraft Co. 연구원. 1991년 ~ 1994년 The Aerospace Corp. 연구원. 1994년 9월 ~ 현재 한국과학기술원 전산학과 조교수.

이 재 원
1992년 2월 한국과학기술원 기계공학과 졸업(학사). 1992년 1월 ~ 현재 (주) 삼성전자 중앙연구소 S/W센터 전임연구원 재직중. 관심분야는 소프트웨어 품질보증, 테스팅 자동화

박 기 웅
1983년 2월 원광대학교 전자공학과 졸업(학사). 1985년 2월 경희대학교 전자공학과 졸업(석사). 1985년 4월 ~ 현재 (주) 삼성전자 중앙연구소 S/W센터 수석연구원 재직중. 관심분야는 소프트웨어 품질보증, 테스팅 자동화, 보안, 전자상거래