韓國컴퓨터情報學會 論文誌
第16卷 第12號, 2011. 12.

2011-16-12-1-1

내장형 소프트웨어 마르코프 체인 모델과 단위 테스트를 이용한 내장형 소프트웨어 신뢰도 분석 도구의 설계와 구현

곽 동 규*, 유 재 우*, 최 재 영*

A Design and Implementation of Reliability Analyzer for Embedded Software using Markov Chain Model and Unit Testing

DongGyu Kwak *, Chae-Woo Yoo *, Jaeyoung Choi *

요 약

내장형 시스템의 요구사항이 복잡해짐에 따라 내장형 소프트웨어의 신뢰도를 분석하기 위한 도구가 요구되고 있다. 소프트웨어의 신뢰도를 분석하는 방법으로는 확률적 모델링을 이용하는데, 다수의 디바이스를 제어하는 내장형 소프트웨어에 적용하기 위해서는 내장형 소프트웨어에 특성화 시킬 필요가 있다. 또한, 기존의 신뢰도 분석 도구는 각 상태간의 전이 확률을 다른 방법으로 측정해야 하고, 한 번 작성한 모델에 대해 재사용을 고려하고 있지 않는다. 본 논문은 내장형 소프트웨어의 신뢰도를 분석하기 위해 내장형 소프트웨어 마르코프 체인 모델과 단위 테스팅 도구를 이용한 신뢰도 분석 도구를 제안한다. 내장형 소프트웨어 마르코프 체인 모델은 신뢰도 분석 방법으로 많이 사용되고 있는 마르코프 체인 모델을 내장형 소프트웨어에 특성화 시킨 모델이다. 그리고 단위 테스팅 도구는 내장형 소프트웨어의 개발환경에 적합한 호스트/타겟 구조를 가지고 있다. 제안하는 도구는 신뢰도 분석을 위해 단위간 전이 확률을 단위 테스트 결과로부터 자동으로 측정하여 기존의 도구보다 용이하게 신뢰도를 분석할 수 있다. 그리고 소프트웨어 모델을 XML 기반의 문서로 표현하여 단위 테스팅 도구가 업데이트 시킨 테스트 결과를 바로 적용할 수 있고, 웹 기반의 인터페이스와 SVN 저장소를 이용하여 다수의 개발자가 쉽게 접근할 수 있는 장점을 갖는다. 본 논문에서는 예제를 이용하여 신뢰도의 분석을 보이고 신뢰도 측정에 유용함을 보인다.

▸ Keyword : 내장형 소프트웨어, 단위 테스트, 신뢰도 분석, 마르코프 체인

Abstract

As requirements of embedded system get complicated, the tool for analyzing the reliability of

• 제1저자 : 곽동규 • 교신저자 : 곽동규
• 투고일 : 2011. 08. 04, 심사일 : 2011. 09. 14, 게재확정일 : 2011. 09. 23.
* 숭실대학교 컴퓨터학부 (School of Computer Science and Engineering)
※ 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 육성지원 사업의 연구결과로 수행되었음
(NIPA-2011-1121-0010)

embedded software is being needed. A probabilistic modeling is used as the way of analyzing the reliability of a software and to apply it to embedded software controlling multiple devices. So, it is necessary to specialize that to embedded software. Also, existing reliability analyzers should measure the transition probability of each condition in different ways and doesn't consider reusing the model once used. In this paper, we suggest a reliability analyzer for embedded software using embedded software Markov chin model and a unit testing tool. Embedded software Markov chain model is model specializing Markov chain model which is used for analyzing reliability to an embedded software. And a unit testing tool has host-target structure which is appropriate to development environment of embedded software. This tool can analyze the reliability more easily than existing tool by automatically measuring the transition probability between units for analyzing reliability from the result of unit testing. It can also directly apply the test result updated by unit testing tool by representing software model as a XML oriented document and has the advantage that many developers can access easily using the web oriented interface and SVN store. In this paper, we show reliability analyzing of a example by so doing show usefulness of reliability analyzer.

▸ Keyword : Embedded Software, Unit Testing, Reliability Analysis, Mark Chain

I. 서 론

최근 내장형 시스템 (Embedded System)의 요구사항이 증가함에 따라 탑재되는 하드웨어 디바이스나 소프트웨어의 복잡도가 증가하고 있다[1]. 이에 따라 신뢰도 높은 내장형 소프트웨어의 개발이 어려워지고 있다. 신뢰도가 높은 소프트웨어란 개발자의 의도에 따라 동작하는 소프트웨어를 의미한다. 소프트웨어의 신뢰도를 분석하는 방법으로는 확률적 모델링을 이용하는 방법[2, 3]이 있는데, 그 중 마르코프 체인 모델 (Markov Chain Modeling)[4]은 단일 행렬과 그 행렬의 승으로 시스템을 분석할 수 있어 표현이 단순하고 계산이 용이하여 시스템을 분석하는데 많이 사용된다[5, 6].

일반적으로 소프트웨어의 신뢰도를 분석하기 위한 마르코프 체인 모델은 사용자의 입력이나 소프트웨어의 분기를 기준으로 상태를 정의하고, 상태간 전이 확률을 측정한 후 전이 행렬을 작성하여 시스템의 신뢰도를 분석한다. 하지만 사용자의 입력을 기준으로 상태를 정의하는 방법은 모든 소프트웨어의 분기를 상태로 나누어 상태의 개수가 많이 신뢰도를 분석하는데 많은 연산이 요구된다. 특히 내장형 소프트웨어의 경우 사용자의 입력이 없는 시스템이 많고, 디바이스 제어 모듈에서 오류가 많이 일어난다. 그러므로 내장형 소프트웨어를 위한 신뢰도 분석 방법은 디바이스 제어 루틴을 고려하여 설계해야 한다.

본 논문은 내장형 소프트웨어의 신뢰도를 분석하기 위하여 내장형 소프트웨어 마르코프 체인 모델을 정의하고, 이를 이용한 신뢰도 분석 도구를 보인다. 내장형 소프트웨어 마르코프 체인 모델은 기존의 마르코프 체인 모델을 내장형 소프트웨어에 특성화 시켜 소프트웨어의 상태를 일반 루틴과 디바이스 제어 루틴을 나누어 정의하고, 이를 이용하여 신뢰도를 분석할 수 있는 방법이다. 내장형 소프트웨어 마르코프 체인 모델은 모든 소프트웨어의 분기로 상태를 나누는 기존의 방법보다 상태의 개수가 적어 신뢰도 분석을 위한 연산의 횟수를 줄이는 장점을 갖는다.

확률적 모델링을 이용한 신뢰도 분석 방법은 자동화된 신뢰도 분석 도구를 적용하여 사용한다[2, 3]. 하지만 기존의 신뢰도 분석 도구는 상태간의 전이 확률을 다른 방법을 이용하여 측정해야 하고, 한번 작성한 시스템 모델을 재사용하기 위한 방법을 제공하고 있지 않는다. 제안하는 신뢰도 분석 도구는 상태간 전이 확률을 자동으로 측정하기 위해 단위 테스팅 도구[7, 8]를 이용하고, 한번 작성한 내장형 소프트웨어 마르코프 체인 모델을 재사용하기 위해 ESM (Embedded Software Markov chain modeling) 문서를 사용한다.

본 신뢰도 분석 도구는 단위 테스팅 도구와 신뢰도 분석기로 구성되어 있는데, 단위 테스팅 도구는 테스트 대상 프로그램과 테스트 케이스를 입력으로 하고 테스트 결과를 출력한다. 그리고 신뢰도 분석기는 테스팅 도구가 생성한 테스트 결과와 ESM 문서를 입력으로 하고 전체 또는 부분 소프트웨어의 신뢰도를 출력한다. 단위 테스팅 도구는 내장형 시스템에서 용이하게 사용할 수 있도록 교차 개발 환경

(Cross-Platform Environment)에 적합한 호스트(Host)/타겟 (Target) 구조를 갖는다. 사용자는 호스트 컴퓨터에서 GUI 기반의 테스트 스크립트 에디터를 이용하여 테스트 케이스를 작성한다. 작성된 테스트 케이스는 테스트 타겟 언어로 변환하여 타겟 컴퓨터에서 실행된다. 그리고 테스트 결과는 호스트 컴퓨터로 전송되고 이는 GUI 기반의 결과 보고서를 생성하여 직관적으로 확인할 수 있다. 또한 테스트 결과는 XML 기반의 문서로서 SVN (Subversion)[9] 저장소 (Repository)를 통해 저장/관리된다. 신뢰도 분석기는 단위 테스팅 도구가 생성한 결과를 바탕으로 각 상태간 전이 확률을 측정한다. 그리고 사용자가 작성한 ESM 문서와 상태간 전이 확률로 전이 행렬을 생성하고 이를 연산하여 전체 시스템의 신뢰도를 분석한다. XML 기반 ESM 문서는 사용자가 한 번 작성한 모델을 저장하여 테스트 결과가 갱신되면 이에 따른 신뢰도를 바로 분석할 수 있다.

제안하는 신뢰도 분석 도구는 웹 기반의 인터페이스를 갖는다. 한 내장형 시스템에서 동작하는 소프트웨어는 다수의 개발자에 의해 작성되는데, 각 개발자는 자신이 작성한 소프트웨어의 신뢰도를 분석할 수 있어야 하고, 소프트웨어를 통합하는 관리자는 모든 단위 소프트웨어의 신뢰도를 분석할 수 있어야 한다. 제안하는 신뢰도 분석 도구는 다수의 개발자가 접근이 용이하여, 테스트 대상 소프트웨어의 마르코프 체인 모델을 한 번 작성하면 다른 개발자가 재사용할 수 있는 장점을 갖는다.

본 논문은 2장에서 기존의 마르코프 체인 모델을 이용한 신뢰도 분석 방법에 관해 논하고 3장에서 내장형 소프트웨어 마르코프 체인 모델과 이를 이용한 신뢰도 분석 방법을 보이며, 4장에서 신뢰도 분석 도구의 구성과 구현을 기술한다. 그리고 5장에서 실험 결과를 보인 후, 6장에서 결론을 맺는다.

II. 신뢰도 분석 도구

비행 제어 시스템은 고장이 발생할 경우 상당한 비용과 인명 등의 피해를 초래하는 분야로서 시스템의 높은 신뢰도가 요구된다. 일반적으로 비행 제어 시스템은 내고장 설계의 한 방법으로 하드웨어를 중복적으로 구성하여 시스템의 신뢰도를 높이고 있다. 중복 구조 신뢰도 분석 도구[10]는 이와 같은 비행 제어 시스템의 신뢰도를 분석하기 위해 하드웨어의 중복 구조를 분류하였다. 그리고 신뢰도 분석 방법을 직렬 시스템과 병렬 시스템, 대기 시스템으로 구분하여 각각의 신로도 분석 방법을 보이고 이를 위한 분석도구를 구현하였다.

다른 신뢰도 분석도구로는 Galileo[3]가 있다. Galileo는 직관적으로 소프트웨어의 구조를 트리 모델로 작성하고 분석할 수 있는 실패 트리 분석 도구이다. 이 도구는 DIFTree (Dynamic Innovative Fault Tree)[11]를 이용하여 패키지 기반 프로그래밍 기법을 사용한 소프트웨어의 신뢰도를 분석한다. 하지만 두 신뢰도 분석 도구 모두 단위의 고장이나 분기 확률을 사용자가 직접 입력해야 한다. 이는 고장이나 분기 확률을 측정하기 위한 다른 방법을 요구한다.

III. 마르코프 체인 모델

마르코프 체인 모델은 상태간의 전이 확률을 바탕으로 다수의 상태를 가지고 있는 시스템에서 한 상태에 있을 확률을 계산하는 방법으로 시스템의 신뢰도를 분석하는데 많이 사용되고 있다[4, 6, 12, 13]. 확률 과정의 하나로 마르코프 과정에서 사용하는 모델인 마르코프 체인은 아래와 같다.

○ 마르코프 체인

$X(t)$가 확률과정일 때, 어떤 n의 $X(n)$가 취하는 값을 상태라 한다. 이 때 t는 우연한 시간 $X(n)=a_i$인 것을 n 단계 (n 회)시행에서 상태 a_i에 있다고 한다. 따라서 확률변수 $X(t)$가 $X_1, X_2, ..., X_n$일 때 대응하는 상태 공간은 유한, 가산 집합 $a_1, a_2, ..., a_n$에 속한다. 상태의 순서쌍 (a_i, a_j)의 확률은 $P_{ij}^{(n)}$으로 나타내고 확률 변수 $X(n), X(n+1)$의 상태가 a_i, a_j일 때, 상태 a_i의 바로 다음 단계의 시행 결과의 확률이 $P_{ij}^{(n)}$이라는 뜻이다. 이러한 추이과정을 마르코프 체인이라 한다.

마르코프 체인은 어떤 시스템을 모형화하는 수학적 기법으로 과거에 있었던 변화를 토대로 시스템의 여러 변수들이 갖고 있는 동적 성격을 파악하여 미래에 있을 변화를 연속적으로 예측하는데 사용되고 있다. 본 논문은 기존의 신뢰도 분석 방법을 보이기 위해 내장형 소프트웨어 중 하나인 LED를 제어하는 예제 소프트웨어를 사용한다. 그림 1은 예제 소프트웨어이다.

```
1 int LED_Control(){
2 int i, LED_fd;  // S1 start
3 unsigned char *mmap_addr;
4 unsigned char *LEDReg;  // S1 end
5 if((LED_fd=open("/dev/mem",O_RDWR|O_SYNC))<0){
   // S2 start end
6 perror("Fail : File Descriptor isn't Opend \n"); // S3
   start
7 exit(2); }  // S3 end
```

```
8 mmap_addr = mmap(NULL,1024,(PROT_READ|PROT_WRITE),
9 MAP_SHARED, LED_fd, 0x08000000); // S4 start end
10 if(mmap_addr < 0) { // S5 start
11 mmap_addr = NULL;  // S6 start
12 printf("Fail : Memory mapping error\n");
13 return -1; }  // S6 end
14 LEDReg = (unsigned char *)(mmap_addr + 8);  //
S7 start
15 i = 0;  // S7 end
16 while(i < 8) {  // S8 start end
17  i++;  // S9 start
18  *LEDReg = 0x << i;
19  sleep(1); } // S9 end
20 munmap(mmap_addr, 1024); // S10 start
21 close(LED_fd);
22 return 0; } // S10 end
23 int main(){
24  int i, n=100;
25  for(i = 0; i < n; i++) {
26   LED_Control();
27  return 0; }
```

그림 1. LED를 컨트롤하는 예제 소프트웨어
Fig. 1. Example Software of LED Control

그림 1의 예제 프로그램을 실행 단위와 조건에 따라 상태로 구분하면 총 10개의 상태로 구성되어 있다. 표 1은 그림 1 소프트웨어의 상태를 나타낸다.

표 1. 예제 소프트웨어의 상태
Table 1. State of Example Software

상태 이름	상태에 해당하는 코드
S1	2 ~ 4줄
S2	5줄
S3	6 ~ 7줄
S4	8줄
S5	9줄
S6	10 ~ 13줄
S7	14 ~ 15줄
S8	16줄
S9	17 ~ 19줄
S10	20 ~ 22줄

예제 프로그램의 상태를 표 1과 같이 나누고 이 상태 전이 그래프를 그리면 그림 2와 같다. 그림 2에서 S3상태와 상태 S6일 때, 소프트웨어는 에러를 발생하게 된다.

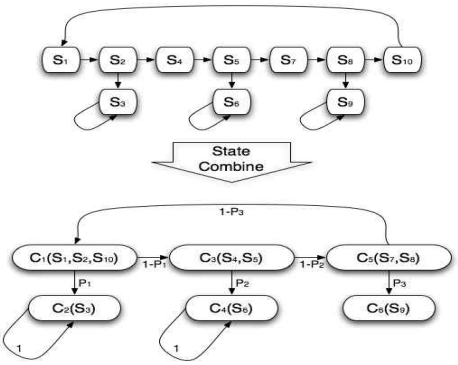

그림 2. 예제 소프트웨어의 상태 전이도와 상태 통합
Fig. 2. State Transition Diagram of Example Software and State Combination

그림 2는 그림 1의 예제 소프트웨어를 분석한 상태 전이도와 상태를 통합한 결과이다. 본 상태 전이도를 이용하여 신뢰도를 분석하기 위해서는 전이 확률을 측정해야 하는데, 전이 확률은 프로그램을 분석하거나 실행하여 측정해야 한다. 확률 값을 바탕으로 연산할 전이 행렬 T를 작성하여 이를 1000회 전이한다고 가정하면 T^{1000}을 계산하면 된다. 아래의 행렬 연산은 T와 T^{1000}이다.

$$T=\begin{bmatrix} 0 & P_1 & 1-P_1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & P_2 & 1-P_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1-P_3 & 0 & 0 & 0 & 0 & P_3 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$=\begin{bmatrix} 0 & 0.01 & 0.99 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & 0.98 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.11 & 0 & 0 & 0 & 0 & 0.89 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$T^{1000}=\begin{bmatrix} 0.01 & 0.267 & 0.011 & 0.528 & 0.097 & 0.087 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0.01 & 0.26 & 0.011 & 0.534 & 0.098 & 0.087 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0.011 & 0.266 & 0.011 & 0.524 & 0.1 & 0.089 \\ 0.011 & 0.266 & 0.011 & 0.524 & 0.1 & 0.089 \end{bmatrix}$$

위 결과에서 S3로 소프트웨어가 종료할 확률은 약 27%, S6로 소프트웨어가 종료할 확률은 약 53%로 분석되고 있다. 본 장에서 보인 마르코프 체인 모델을 이용한 신뢰도 분석 방법은 상태간의 전이 확률을 이용하여 전체 소프트웨어의 신뢰도를 측정한다. 하지만 이 방법은 디바이스를 제어하는 상태와 일반 상태를 구분하고 있지 않아 상태가 많아 전이가 많이 일어나는 시스템일 경우 많은 연산을 요구한다. 본 논문은 상태를 줄이기 위해서 일반 루틴과 디바이스 제어 루틴을 다른

형태로 다루어 디바이스 제어 루틴의 에러에 따른 내장형 시스템의 신뢰도를 측정한다.

IV. 내장형 소프트웨어 마르코프 체인 모델

본 논문의 목적은 마르코프 체인 모델을 특성화 시킨 내장형 소프트웨어 마르코프 체인 모델을 이용하여 내장형 소프트웨어의 신뢰도를 분석하는데 있다. 마르코프 체인 모델은 전체 시스템을 상태간의 전이로 모델링하여 시스템이 한 상태에서 있을 확률을 분석하는 방법이다. 본 장에서는 내장형 소프트웨어의 신뢰도를 분석하기 위해 마르코프 체인을 내장형 소프트웨어에 특성화 시켜 내장형 소프트웨어 마르코프 체인 모델을 정의한다.

○ 내장형 소프트웨어 마르코프 체인 모델

상태 집합 $S = N \cup D \cup E$

(단, N은 일반 루틴, D는 디바이스 제어 루틴, E는 에러 핸들링 루틴, $N \cap D = \Phi, D \cap E = \Phi, E \cap N = \Phi$)

$S(t)$가 확률과정일 때, $S(t)$가 취하는 값을 상태라 한다. $S(n) = s_i$일 때, 순서쌍 (s_i, s_j)는 s_i상태에서 s_j상태로 전이하는 확률이고 $P_{ij}^{(n)}$으로 표현한다.

일반적으로 내장형 소프트웨어는 디바이스를 컨트롤하는 경우가 많다. 내장형 소프트웨어는 일반 루틴과 디바이스를 제어하는 루틴, 에러 핸들링 루틴으로 구성되어 있다. 내장형 소프트웨어 마르코프 체인 모델은 3 가지 루틴으로 구성되어 있어 내장형 소프트웨어를 분석하기 적합하다. 표 2는 그림 1의 예제를 내장형 소프트웨어 마르코프 체인 모델로 분석한 상태이다.

내장형 소프트웨어 마르코프 체인 모델에서의 상태	이전 모델에서의 상태	상태에 해당하는 코드
N1	S1	2 ~ 4줄
D1	S2	5줄
E1	S3	6 ~ 7줄
D2	S4	8줄
	S5	9줄
E2	S6	10 ~ 13줄
N2	S7	14 ~ 15줄
	S8	16줄
	S9	17 ~ 19줄
	S10	20 ~ 22줄

표 2. 내장형 소프트웨어 마르코프 체인 모델의 상태
Table 2. State of Embedded Software Markov Chain Model

표 2의 상태에 따라 예제 소프트웨어의 상태 전이를 그래프로 표현하고 이를 분석하여 상태를 통합하면 그림 3과 같다.

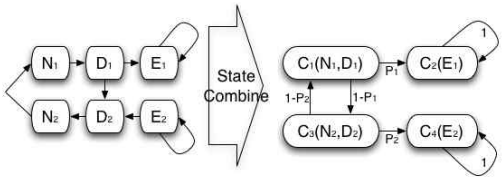

그림 3. 내장형 소프트웨어 마르코프 체인 모델의 상태 전이도와 상태 통합
Fig. 3. State Transition Diagram of Embedded Software Markov Chain Model and State Combination

그림 3의 상태 전이도를 바탕으로 전이 행렬 T를 작성하고 이를 100회 전이한다고 가정하면 T^{100}을 계산하면 된다. 아래의 행렬은 T와 T^{100}이다.

$$T = \begin{bmatrix} 0 & P_1 & 1-P_1 & 0 \\ 0 & 1 & 0 & 0 \\ 1-P_2 & 0 & 0 & P_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.01 & 0.99 & 0 \\ 0 & 1 & 0 & 0 \\ 0.98 & 0 & 0 & 0.02 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T^{100} = \begin{bmatrix} 0 & 0.26 & 0.22 & 0.52 \\ 0 & 1 & 0 & 0 \\ 0.22 & 0.26 & 0 & 0.52 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

위 결과에서 E1으로 소프트웨어가 종료할 확률은 약 26%, E2로 종료될 확률은 52%로 분석되고 있다. 그림 1의 소프트웨어는 전이 횟수를 증가시킬수록 에러 핸들링 루틴에서 종료할 확률이 증가하게 되는데 $P_{12}^{(\infty)} \approx 33.56\%$이고 $P_{14}^{(\infty)} \approx 66.44\%$ 이다. 그림 4는 기존의 마르코프 체인 모델을 이용한 방법 (기존 방법)과 제안하는 내장형 소프트웨어 마르코프 체인 모델 (제안 방법)의 전이횟수의 증가에 따른 상태 확률을 보여준다.

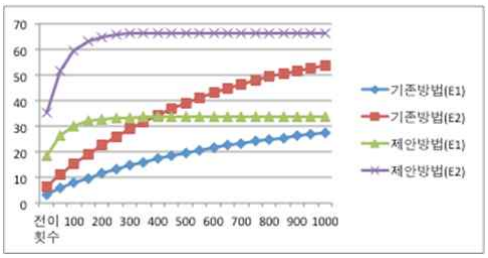

그림 4. 전이 횟수와 상태 확률
Fig. 4. Number of Transition and State Probability

그림 4와 같이 상태 확률은 기존의 방법에 비해 제안하는 방법이 빠르게 수렴하고 있다. 마르코프 체인 모델에서의 상태는 분기가 일어나는 위치에서 상태가 나누어지는데, 내장형 소프트웨어 마르코프 체인 모델에서는 디바이스를 제어하는 루틴에서의 분기를 기준으로 상태를 나누어 일반 루틴에서 발생하는 분기는 상태로 나누지 않아 상태에 개수가 줄어든다. 그러므로 내장형 소프트웨어 마르코프 체인 모델은 신뢰도를 분석하기 위한 복잡도와 연산의 횟수를 줄일 수 있다.

V. 신뢰도 분석도구

제안하는 신뢰도 분석 도구는 단위 테스트 도구와 내장형 소프트웨어 마르코프 체인 모델을 이용한 신뢰도 분석 도구이다. 내장형 소프트웨어 마르코프 체인 모델을 이용하여 전체 시스템의 신뢰도를 분석하기 위해서는 각 상태간 전이 확률을 측정해야 한다. 본 신뢰도 분석 도구는 단위 테스팅 도구를 이용하여 각 상태간 전이 확률을 측정하고, 이를 이용하여 전체 시스템의 신뢰도를 분석한다. 그림 5는 신뢰도 분석 도구의 전체 구조이다.

단위 모듈 개발자는 자신이 개발한 모듈과 전이 관계가 있는 모듈과의 테스트를 실시하기 위한 테스트 케이스를 작성한다. 단위 테스팅 도구는 단위 모듈 개발자가 작성한 테스트 케이스와 테스트 대상 소스로 테스트를 실행하고 이는 테스트 결과 저장소에 저장된다. 그리고 소프트웨어 전체의 신뢰도를 분석하는 개발자는 소프트웨어의 내장형 소프트웨어 마르코프 체인 문서를 작성하고 신뢰도 분석 도구를 이용하여 신뢰도를 분석한다. 신뢰도 분석 도구는 내장형 소프트웨어 마르코프 체인 문서와 테스트 결과를 입력으로 하고, 신뢰도를 출력으로 한다.

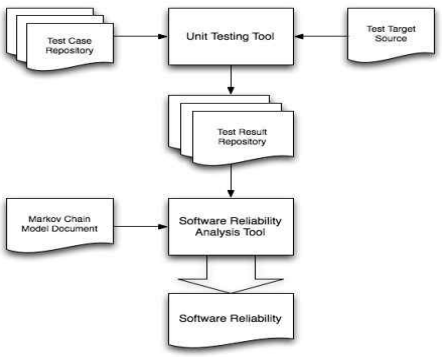

그림 5. 신뢰도 분석 도구의 시스템 구조
Fig. 5. System Architecture of Reliability Analyzer

4.1 확률 측정기

본 논문은 신뢰도를 분석하기 위해 단위 테스팅 도구의 결과를 이용하여 각 상태간 전이 확률을 측정한다. 단위 테스팅 도구는 테스트 케이스와 대상 소프트웨어를 입력으로 하고 테스트 결과를 생성한다. 테스트 케이스는 대상 소프트웨어의 입력과 소프트웨어의 예상 출력을 포함한다. 그리고 테스트 결과는 테스트 케이스의 예상 결과와 동일할 경우 성공을 다를 경우 실패를 의미한다. 그림 6은 예제 프로그램을 단위 테스팅 도구로 테스트하여 생성한 테스트 결과이다.

```
<testResult><test name="ss_mmap_1" result="success" >
  <element runTime="0" result="success" >
    <var name="length" init="1024" ev="" initResult="1"
evResult=""
     result="success" />
    <var name="prot" init="1" ev="" initResult="1"
evResult=""
     result="success" /> ... </element>
 </test>
 <test><element runTime="0" result="fail">
    <var name="length" init="1024" ev="" initResult="1024"
evResult=""
     result="success" />
    <var name="prot" init="1" ev="" initResult="1"
evResult=""
     result="success" /> ... </element>
 </test> ...
</testResult>
```

그림 6. 확률 측정을 위한 테스트 결과 XML
Fig. 6. Test Result XML of Probability

단위 테스팅 도구가 생성한 테스트 결과는 확률 분석기를 통해 상태간 전이 확률을 측정한다. 확률 측정기는 테스트 결과 파서와 확률 계산기로 구성되어 있다. 테스트 결과 파서는 XML 기반의 테스트 결과 문서를 입력으로 받아 파싱하여 테스트 결과를 추출하고 확률 계산기에 전달한다. 아래는 확률 계산 알고리즘이다.

○ 확률 계산 알고리즘

SUCCESS : 전이한 테스트 결과

FAIL : 전이하지 않은 테스트 결과

```
probabilityCalculator(testResult) {
  int successCounter = 0;
  int failCounter = 0;
  for(int i = 0; i < testResult 배열의 개수; i++){
    switch(testResult[i].result){ // 테스트 결과 확인
      case SUCCESS :
        successCounter++; // 전이한 경우 카운트
      case FAIL :
```

failCounter++; // 전이하지 않은 경우 카운트 }
probability = successCounter / testResult 배열의 개수; // 확률 계산 }}

확률 계산 알고리즘은 한 상태에서 다른 상태로 전이하는 확률을 테스트 결과를 이용하여 계산한다. 테스트 결과는 단위 테스팅 도구에 의해 생성되는데, 단위 테스팅 도구는 단위 개발자가 입력에 따른 예상 결과를 실제 단위 소프트웨어를 실행시켜 확인할 수 있다. 단위 개발자는 테스트 케이스를 작성하는데 테스트 케이스에는 한 상태에서 다른 상태로 전이할 경우를 예상하고 있다. 그리고 실행 결과가 예상과 동일할 경우 성공으로 표현하고 다른 경우는 실패로 표현한다. 그러므로 단위 테스트 결과의 성공과 실패를 이용하여 전이 확률을 계산할 수 있다. 확률 계산 알고리즘은 다수의 테스트 결과를 입력으로 받아 성공과 실패를 카운트하여 전이 확률을 계산한다. 계산된 전이 확률은 내장형 소프트웨어 마르코프 체인 모델과 함께 신뢰도를 분석하는데 사용된다.

4.2 신뢰도 분석기

제안하는 신뢰도 분석 도구는 단위 테스팅 도구의 결과를 이용하여 각 상태간 전이 확률을 생성하고, 내장형 소프트웨어 마르코프 체인 모델을 이용하여 신뢰도를 분석한다. 본 논문은 내장형 소프트웨어 마르코프 체인 모델을 XML 문서로 작성할 수 있는 ESM (Embedded Software Markov chain modeling) 문서를 제안한다. ESM 문서는 한 번 작성한 내장형 소프트웨어 마르코프 체인 모델을 제사용하고 갱신되는 테스트 결과를 바로 적용할 수 있는 장점을 가진다. 그림 7은 그림 1의 예제 소프트웨어를 분석하여 작성한 ESM 문서이다.

```
<markov_model><states>
<state name="n1" type="stable"><section start="9"
end="11"/></state>
<state name="n2" type="stable"><section start="23"
end="24"/></state>
... <function name="mmap" /></state>...
<state name="n1d1" type="risk"><substate name="d2"/>
<substate name="n2"/><state></states>
<transitions><transition state="n1d1" to="e1"
probability="unknow"/>
<transition state="n1d1" to="d2n2"
probability="unknow"/>
<transition state="d2n2" to="e2"
probability="unknow"/>
...</transitions></markov_model
```

그림 7. 내장형 소프트웨어 마르코프 체인 모델 문서
Fig. 7. Document of Embedded Software Markov Chain Modeling

ESM 문서는 프로그램을 라인 단위로 분류하여 상태를 정의한다. 그리고 전이가 일어나는 상태들을 “<transition>” 태그로 기술한다. 그림 7은 그림 3의 내장형 소프트웨어 마르코프 체인 모델이 가지고 있는 정보를 모두 포함하고 있다. 신뢰도 분석기는 ESM 문서를 이용하여 신뢰도를 분석한다. ESM 문서는 단위 테스팅 도구가 생성한 테스트 결과가 추가되어 상태간 전이 확률이 갱신될 경우 갱신된 결과를 바로 신뢰도에 적용할 수 있도록 한다. 아래는 신뢰도를 분석하는 알고리즘이다.

○ 신뢰도 분석 알고리즘
N : 상태의 개수
ST : 상태 전이 횟수
입력 : transitionProbability[][] / 상태간 전이 확률
출력 : stateProbability[][] / 전체 소프트웨어 상태 확률

```
reliabilityCalculator(transitionProbability) {
  stateProbability = transitionProbability;
  for(int i = 0; i < ST; i++) {
    stateProbability = mathMultiplication(stateProbability,
transitionProbability); // 상태전이 횟수만큼 전이 행렬을 곱}
  return stateProbability; // 전체 소프트웨어의 상태 확인}
matrixMultiplication(matrix1, matrix2) { // 두 행렬의 곱
  resultMatrix[][];
  for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++){
      for(int k = 0; k < N; k++){
        resultMatrix[i][j] = resultMatrix[i][j] +
(matrix1[i][k] * matrix[k][j]);  }}}
  return resultMatrix; // 곱한 결과를 반환}
```

신뢰도 분석 알고리즘은 시스템이 ST회 전이가 일어났을 때, 각 상태에 있을 확률을 계산한다. 신뢰도 분석 알고리즘의 입력은 상태간 전이 확률이고 출력은 소프트웨어의 상태 확률이다. 전이 확률은 전이 행렬로 표현되고 마르코프 체인 모델의 전이 확률 계산 방법에 따라 행렬의 승을 계산하여 전체 소프트웨어의 상태 확률을 측정한다. 즉, 신뢰도 측정 알고리즘은 전이 행렬의 ST승을 계산한다. 전체 소프트웨어의 상태 확률을 표현하는 결과 행렬 중 시작 상태에 해당하는 행을 분석하면 시스템의 신뢰도를 분석할 수 있다.

VI. 실험

본 장은 제안하는 신뢰도 분석 도구를 이용하여 그림 1에

서 보인 소프트웨어의 신뢰도를 분석한다. 그리고 단위 테스팅 도구의 스크립트를 이용하여 전체 소프트웨어를 테스트하고 두 가지 방법에서 생성한 신뢰도를 비교 분석한다. 단위 테스팅 도구를 이용한 신뢰도 분석 방법은 전체 소프트웨어에 대한 테스트 스크립트를 작성하여 테스트를 실시하고, 테스트 결과를 직접 통계 내어 계산한다. 그리고 제안하는 신뢰도 분석 도구를 이용한 방법은 각 단위에 대한 테스트 스크립트를 작성하여 단위의 테스트 결과를 생성한다. 생성된 테스트 결과를 이용하여 전이 확률을 계산하고, 전이 확률을 내장형 소프트웨어 마르코프 체인 모델에 적용한다. 본 도구는 마르코프 체인 모델과 전이 확률로 전이 행렬을 생성하고 이를 연산하여 소프트웨어의 신뢰도를 분석한다. 표 3은 그림 3의 내장형 소프트웨어 마르코프 체인 모델에서 P1과 P2를 단위 테스팅 도구를 이용하여 측정한 결과이다.

표 3. 전이 확률
Table 3. Transition Probability

변수	실행횟수	정상 전이된 횟수	에러 전이된 횟수	정상 확률 값
P1	100	99	1	0.99
P2	100	98	2	0.98

표 3의 전이 확률은 ESM 문서의 “<transition>” 태그의 “probability” 속성에 기록하고 테스트 결과가 업데이트되어 전이 확률이 변경될 경우 수정한다. 신뢰도 분석기는 ESM 문서를 바탕으로 신뢰도를 분석하여 그래프와 표로 결과를 생성한다. 그림 8은 신뢰도 분석 결과이다.

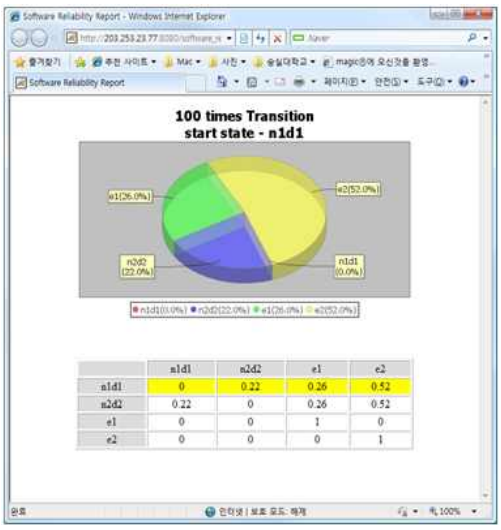

그림 8. 신뢰도 분석 결과 화면
Fig. 8. Result Page of Reliability Analyzer

그림 8과 같이 분석된 결과인 실제 소프트웨어를 실행한 결과와 비교하여 제안하는 도구를 검증한다. 표 4는 소프트웨어를 실제로 실행하여 분석한 신뢰도 결과이다.

표 4. 신뢰도 분석 도구의 분석 값
Table 4. Value by Reliability Analyzer

실행 횟수	정상으로 종료된 횟수	에러로 종료된 횟수	정상 상태로 종료
100	22	78	0.22

제안하는 분석도구와 실행을 통해 신뢰도를 분석하여 유사한 분석 결과를 얻을 수 있음을 확인할 수 있다. 본 논문은 위와 같은 실험을 다섯 개의 내장형 소프트웨어에 적용하여 제안하는 신뢰도 분석 도구를 검증한다. 표 5는 제안하는 도구를 이용하여 분석한 소프트웨어의 신뢰도와 실행을 시켜 얻은 신뢰도이다.

표 5. 신뢰도 분석 도구의 분석 값
Table 5. Value by Reliability Analyzer

소프트웨어	제안하는 도구	실행을 통한 장상 종료 백분율	오차
A	96%	100% (50/50)	5%
B	34%	36% (18/50)	2%
C	2%	3% (3/100)	1%
D	88%	88% (88/100)	0%
E	22%	22% (22/100)	0%

표 5는 제안하는 도구를 이용하여 신뢰도를 분석한 결과와 소프트웨어를 실행시켜 얻은 정상 종료 백분율이다. 신뢰도 분석 도구로 분석한 신뢰도와 실험을 통한 정상 종료 횟수의 오차는 5%이하이다. 그리고 이 오차는 실행 횟수가 증가할수록 작아진다.

Ⅶ. 결 론

내장형 시스템의 요구사항이 증가함에 따라 신뢰도가 높은 소프트웨어의 작성이 어려워지고 있다. 이에 따라 내장형 소프트웨어의 신뢰도를 분석하기 위한 방법과 도구가 요구된다. 시스템의 신뢰도를 분석하는 방법으로는 확률적 모델을 이용한 방법이 많이 사용되고 있고, 이를 적용한 도구가 연구되었다. 하지만 기존의 확률적 모델은 주로 디바이스 제어를 통해 요구사항을 만족 시키는 내장형 소프트웨어에 적합하지 않아 적용하기 어렵다. 또한, 신뢰도 분석 도구는 각 상태간의 전이 확률을 다른 방법으로 측정하여 입력해야 한다.

본 논문은 내장형 소프트웨어 마르코프 체인 모델링 방법

과 단위 테스팅 도구를 이용한 내장형 소프트웨어 신뢰도 분석 도구를 제안한다. 내장형 소프트웨어 마르코프 체인 모델링 방법은 확률적 모델링 방법 중 시스템 분석에 많이 사용되는 마르코프 체인 모델을 내장형 소프트웨어에 특성화 시킨 모델이다. 내장형 소프트웨어는 장치한 디바이스를 제어하는 특성을 갖는다. 그리고 단위 테스팅 도구는 테스트 대상이 되는 소프트웨어의 예상 결과를 테스트 스크립트로 입력하고 대상 소프트웨어를 실행 시켜 소프트웨어가 예상 결과와 동일하게 동작하는지를 테스트하는 도구이다. 제안하는 신뢰도 분석 도구는 단위 테스팅 도구가 생성한 테스트 결과로 상태간 전이 확률을 측정하고 이를 전이 행렬에 적용하여 신뢰도를 분석한다.

본 신뢰도 분석 도구는 단위 테스팅 도구와 신뢰도 분석기로 구성되어 있다. 단위 테스팅 도구는 내장형 개발 환경에 용이하게 사용하도록 호스트/타겟 구조를 갖는다. 테스트를 실시하는 개발자는 호스트에서 테스트 스크립트를 작성하고 이를 타겟에서 실행한다. 그리고 테스트 결과를 호스트에서 그래프와 표를 통해 직관적으로 확인할 수 있고, 이는 SVN 저장소를 통해 관리된다. 신뢰도 분석기 중 일부인 확률 측정기는 테스트 결과를 분석하여 각 상태간 전이 확률을 측정한다. 그리고 신뢰도 분석기는 사용자가 작성한 EMS 문서와 각 상태간 전이 확률을 이용하여 전체 소프트웨어의 신뢰도를 분석한다. EMS는 소프트웨어의 내장형 소프트웨어 마르코프 체인 모델에 관한 정보를 가지고 있는 XML 기반의 문서이다. 제안하는 신뢰도 분석 도구는 한 번 작성한 EMS 문서를 이용하여 테스트 결과에 따른 확률의 갱신을 신뢰도에 바로 적용할 수 있는 장점을 갖는다. 그리고 SVN 저장소와 웹 기반의 인터페이스를 이용하여 다수의 개발자가 쉽게 접근하여 테스트 결과를 관리하고 신뢰도 측정 결과를 확인할 수 있다.

참고문헌

[1] B. Beizer, Software Testing Techniques. Van Nost rand Reinhold 2nd edition, 1990.

[2] Sung-su Kim, Sanghyuk Park, Sung-Hwan Kim, Keeyoung Choi, Cheol-Keun Ha, Choon-Bae Park, “Development of Reliability Block Diagram Analysis Tool for H/W Redundancy Structure based Unit Module”, Journal of KSAS v.37 n.6, pp. 595~601, 2009. 6.

[3] Kevin J. Sullivan, Joanne Bechta Dugan, David Coppit, “The Galileo Fault Tree Analysis Tool”, Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Comp uting, p. 232, June 15~18, 1999.

[4] Eun-Gu Lee, “A Study on Markov Chain”, Journal of Korea Society of Mathematical Education v.20 n.3, pp. 19~22, 1982. 6.

[5] Gwendolyn H. Walton, J. H. Poore, “Generating transition probabilities to support model-based software testing”, Software-Practice & Experience v.30 n.10, pp. 19~22, 1982. 6.

[6] Janes A. Whittaker, Michaei G. Thomason, “A Markov Chain Model for Statistical Software Testing”, IEEE Transactions on Software Engineering v.20 n.10, pp. 812~824, October. 1994.

[7] Donggyu Kwak, Chae-Woo Yoo, Yongyun Cho, “A Software Unit Testing Tool based on The XML Test Script for Embedded Systems”, Journal of Korea Society of Computer and Information v.14 n.1, pp. 17~24, 2009. 1.

[8] Donggyu Kwak, Chae-Woo Yoo, Yongyun Cho “A Software Unit Testing Tool based on The XML Test Script for Embedded System”, Journal of the Korea Society of Computing & Information, v.14, n.1, pp. 17 ~ 24, 2009. 1.

[9] Subversion, http://subversion.tirgris.org.

[10] Sung-su Kim, Sanghyuk Park, “Development of Reliability Block Diagram Analysis Tool for H/W Redundancy Structure based on Unit Module”, Journal of the Korean Society for Aeronautical & Space Sciences, v.37 n.6, pp. 595~601, 2009. 6.

[11] J. Dugan, K. Venkataraman, R. Gulati, "DIFtree: A software package for the analysis of dynamic fault tree models", Proc. 1997 Reliability and Maintainability Symposium, January 1997.

[12] J. Rajgopal, M. Mazumdar, “Modular operational test plans for inferences on software reliability based on a Markov model”, Software Engineering,

IEEE Transactions v. 28, pp. 358~363, 2002. 4.
[13] Jeffery Horn, "Finite Markov Chain Analysis of Genetic Algorithms with Niching", Proceedings of the 5th International Conference on Genetic Algorithms, pp. 110~117, 1993.
[14] Kim Hee Cheul, " The Study for NHPP Software Reliability Growth Model based on Exponentiated Exponential Distribution", Journal of the Korea Society of Computing & Information, v.11, n.5, pp. 9 ~ 18, 2006. 11.

저 자 소 개

곽 동 규

2002 : 서경대학교 응용수학과 이학사.
2004 : 숭실대학교 컴퓨터학과 공학석사.
현 재 : 숭실대학교 컴퓨터학부 박사과정
관심분야 : 프로그래밍 언어, 컴파일러, XML, 임베디드 시스템, 유비쿼터스
Email : coolman@ss.ssu.ac.kr

유 재 우

1976 : 숭실대학교 전자계산학과 이학사.
1985 : 한국과학기술원 전산학과 공학박사.
1986 : 코넬대학교 객원교수
1999 : 한국정보과학회 프로그래밍 언어 연구회 위원장
현 재 : 숭실대학교 컴퓨터학부 교수
관심분야 : 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용
Email : cwyoo@ssu.ac.kr

최 재 영

1984 : 서울대학교 제어계측공학과 (학사)
1986 : 미국 남가주대학교 컴퓨터공학 공학석사.
1991 : 미국 코넬대학교 컴퓨터공학 공학박사
1994 : 미국 국립 오크리지연수소 연구원
1995 : 미국 테네시 주립대학교 연구교수
현 재 : 숭실대학교 컴퓨터학부 교수
관심분야 : 시스템소프트웨어, 고성능 컴퓨팅(HPC), 유비쿼터스 컴퓨팅
Email : choi@ssu.ac.kr