□특 집□

결함 허용 소프트웨어의 신뢰도 모델링 기법

숭실대학교 소프트웨어공학과 양 승 민*

목 차

I. 서 론

신뢰도 높은 소프트웨어 개발의 필요성은 전혀 새로운 것이 아니다. 컴퓨터가 만들어진 이후 특히 1970년대 이후 소프트웨어의 중요성에 대한 인식과 함께 결함 없는 소프트웨어의 개발에 대한 많은 연구가 있었다[8]. 그 중에서도 formal method에 의한 명세 및 correctness 증명 방법 그리고 테스팅이 대표적이다[10]. 그러나 테스팅은 오류의 존재는 보여줄 수 있지만 오류의 부재는 보장할 수 없으므로 테스팅이 완료되었다는 표시가 없다. 그리고 correctness 증명은 상위 레벨 설계시나 프로그램 일부분에 대해서는 가능하지만 그 작업이 쉽지 않을 뿐더러 커다란 소프트웨어에 대한 증명은 현실적으로 불가능하며 증명시의 오류발생도 가능하다. 더구나 옳은 프로그램이라도 나쁜 데이타나 비정상적인 수행환경 그리고 하드웨어에의 결함에 의해 옳지 않은 결과를 산출할 수 있다. 따라서 고신뢰도를 요구하는 시스템의 소프트웨어는 개발시 결함을 없애는 노력과 더불어 결함 허용을 하는 소프트웨어 즉 수행시 어떤 부분에 결함이 발생하였을 때라도 올바른 결과를 산출할 수 있는 방법이 고려되어야 한다. 이러한 소프트웨어를 결함 허용 소프트웨어라 한다.

지난 이십년간 여러 종류의 결함 허용 소프트웨어의 개발을 위한 연구가 있었다[1,5,12]. (자세한 것은 본 특집의 "소프트웨어 결함 허용 기법에 대한 고찰"참조) 예를 들어 중복 소프트웨어의 개발, 결함 탐지 방법, 결함 복구를 위한 소프트웨어의 구조에 관한 연구이다. 이와 더불어 이러한 결함 허용 소프트웨어의 신뢰도 측정(또는 예측)에 대한 연구 또한 매우 중요하다. 각 요소들의 신뢰도 측정은 물론 시스템 레벨에서의 신뢰도 측정이 가능하여야 하며 이를 위한 신뢰도 모델의 개발이 필수적이다[4,13]. 신뢰도 모델은 다음과 같이 유용하게 사용된다.

- 시스템 신뢰도 요구 사항에 대한 해석 및 결정
- 서로 다른 시스템 형태(configuration)에 대한 신뢰도 예측

* 종신회원

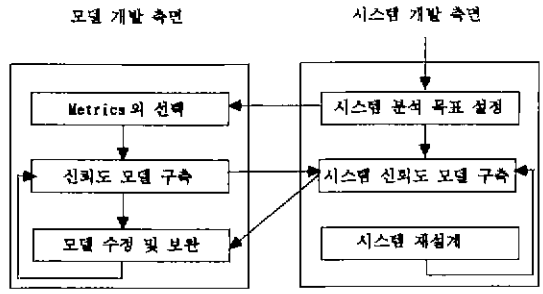
－시스템 개발시 신뢰도 약점 식별
－효과적인 운용 및 유지 보수 계획 수립

본 논문은 결함 허용 소프트웨어의 신뢰도를 측정하기 위한 모델링 기법에 대해 소개한다. 결함 허용 기법중 가장 대표적인 복구 블럭(recovery block)과 N 버전 프로그래밍(NVP)을 중심으로 신뢰도 모델링 기법에 대해 알아보고 앞으로의 연구 방향에 대해 논의한다. 우선 다음 장에서는 전반적인 신뢰도 모델링 프로세스에 관해 알아본다.

II. 신뢰도 모델링 프로세스

신뢰도 모델은 시스템 개발시 여러 단계에서 사용될 수 있으며 궁극적인 목적은 적절한 비용으로 주어진 신뢰도를 만족하는 시스템을 개발하고자 함이다. 상위 레벨 설계시의 신뢰도 모델은 시스템의 신뢰도를 만족시키기 위한 서브 시스템의 신뢰도 할당(reliability budget)을 하는데 사용할 수 있다. 즉 시스템에 요구된 신뢰도를 만족시켜 주기 위하여 각 서브 시스템의 신뢰도에 대한 요구를 구할 수 있다. 그리고 서브 시스템의 개발 후 각 서브 시스템의 신뢰도가 구해지면 상위 레벨 신뢰도 모델에 적용함으로써 시스템의 신뢰도를 예측할 수 있고 필요한 경우 신뢰도의 병목(reliability bottleneck)되는 부분에 대한 재설계를 할 수 있다. 이러한 과정을 시스템 설계시 매 단계마다 적용할 수 있다. 즉 한 단계의 모델은 그 다음 단계 요소에 요구되는 신뢰도를 구할 수 있고 그 다음 단계의 구현 후 적용함으로써 요소에 대한 재설계나 모델 개량을 할 수 있다.

이러한 모델링 작업은 그림 1과 같이 크게 두 가지 측면에서 이루어져야 한다. 하나는 모델의 개발과 그에 대한 개량 작업이고 다른 하나는 그 모델을 이용한 시스템의 신뢰도 분석과 그에 따른 시스템의 재설계 작업이다[13]. 우선 시스템 분석 목표를 설정하고 그에 따른 적절한 metrics를 선택한다. 그리고 metrics의 산출을 위한 신뢰도 모델을 구축하고 개발 중인 시스템을 그 모델에 적용한다. 분석 결과에 따라 신뢰도 할당이나 시스템 재설계가 이루어져야 하며 경우에



(그림 1) 신뢰도 모델링 작업도

따라서는 모델 자체의 수정 또는 보완이 필요하다. 대부분의 경우 이러한 작업이 반복적으로 수행된다. 본 장에서는 모델 개발 측면, 즉 다음 세가지 작업에 대해 논의한다.

(1) 시스템 설계 목표에 기초하여 신뢰도 분석을 위한 metrics의 선택;
(2) 시스템 신뢰도 모델 구축; 그리고
(3) 모델에 대한 분석 및 수정.

2.1 분석을 위한 metrics의 선택

시스템의 신뢰도를 평가하기 위한 첫번째 단계는 분석을 위한 metrics를 정하는 것이다. 이 metrics는 시스템의 요구 사항과 특성에 의한 시스템 분석 목표를 정확히 반영할 수 있게 선택되어야 한다. 일반적으로 많이 사용되는 metrics로는 신뢰도(reliability), MTTF(Mean Time To Failure), 유용도(availability), 그리고 예상 실패수(expected number of failure) 등이 있다.

신뢰도 : 신뢰도는 시스템이 주어진 실행 환경하에서 일정기간 동안 정확히 작업을 수행할 수 있는 확률이다[6]. 따라서 시스템 신뢰도는 운용시간이 고려되어야 한다. 어떤 시스템은 몇 달 또는 몇년간 계속 운용되며(예를 들어 교환기 시스템) 어떤 시스템은 몇 시간만 계속 운용된다(예를 들어 비행기 제어 시스템).

MTTF : MTTF는 시스템 실패가 발생할 때까지의 예상 시간이다. 신뢰도와는 달리 운용시간에 의한 함수가 아니고 시스템이 실패없이 운용되는 평균시간을 나타낸다. MTTF는 여러 개의 서로 다른 설계를 비교할 때 쉽게 사용할 수 있는 파라미터이다. 하지만 시스템의 여러

변수를 반영하기가 쉽지 않으므로 소프트웨어 신뢰도 측정을 위해서는 적합하지 않다. 특히 MTTF가 주어진 시스템 운용시간 보다 훨씬 길더라도 그것이 어떤 특정 운용 시간 동안 고신뢰도를 보장한다고 볼 수 없다.

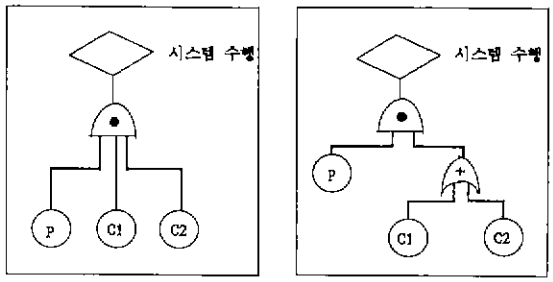
유용도 : 유용도는 시스템이 어떤 주어진 시간에 운용중일 확률이다. 유용도는 주로 주어진 작업 기간안에 사용불능의 상태 기간으로 표시된다(예를 들어 1년에 한시간의 사용 불능). 따라서 통신 스위치 시스템 등은 신뢰도보다 유용도가 더 적합한 metrics이다. 그러나 유용도는 얼마나 자주 사용불능이 되는가는 반영되지 않는다.

실패 예상수 : 실패 예상수는 주어진 운용 시간내에 발생 예상되는 실패의 갯수이다. 이것은 예비품의 수와 보수 능력의 정도를 예측하는데 이용할 수 있으며 하드웨어 요소의 신뢰도 metrics로 적합하다. 그러나 시스템 요소의 실패 예상수와 시스템 레벨에서의 실패 예상수는 꼭 비례하지 않는다. 특히 결함 허용 시스템에서는 요소의 실패 예상수는 증가하나 시스템 전체의 실패 예상수는 감소한다.

이상의 metrics 외에도 여러가지의 metrics가 가능하며 시스템의 특성과 분석 목적에 따라 선택되어야 한다. 그리고 실시간 시스템에서는 신뢰도와 함께 실시간성을 분석하기 위한 metrics도 함께 고려되어야 한다.

2.2 신뢰도 모델 구축

신뢰도 모델은 시스템의 특성 그리고 신뢰도 metrics에 따라 정해져야 한다. 특히 결함 허용 시스템은 다음 두 가지 요인을 추가로 고려하여야 한다. 첫째는 시스템의 요소의 결함 발생시 그에 대한 보상(coverage) 능력이다. 결함 허용 시스템에서는 어떤 요소의 결함 발생시 여분의 요소(redundant component)에 의해 복구가 되는데 이때 100% 복구는 보장 못한다. 그 이유로는 설계 오류로 인한 요소들 간의 공통적인 결함, 결함 탐지의 실패, 그리고 외적인 요소에 의한 결함 등이다. 따라서 결함 발생시 그것을 보상할 수 있는 확률이 신뢰도 모델에 반영되어



a. 직렬 시스템의 성공 트리 b. 병렬 시스템의 성공 트리

P : 전원 C1, C2 : 컴퓨터

(그림 2) 시스템 수행을 위한 성공 트리

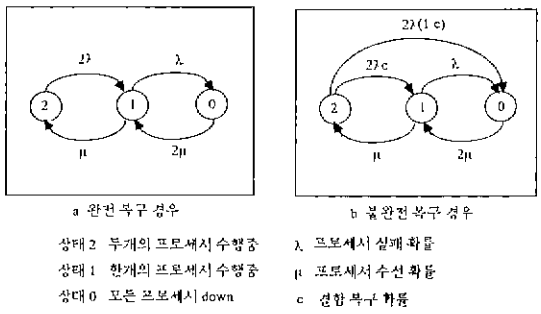
야 한다. 둘째는 주요소와 여분 요소간의 종속성 문제이다. 특히 소프트웨어 요소인 경우 비록 설계가 독립적으로 되었다고 할지라도 주어진 입력에 대해 전혀 독립적인 결함이 발생된다고 보기 어렵다[1]. 따라서 요소간의 종속 확률에 대한 것도 고려되어야 한다.

가장 많이 사용되는 신뢰도 모델로는 부품 계수(part-count)모델, 콤비나토리알(combinatorial)모델, 그리고 상태-공간(state-space) 모델이다.

부품 계수 모델 : 가장 간단한 모델링 방법으로서 시스템의 신뢰도를 첫 눈에 예측할 수 있는 방법이다. 이 방법에서는 시스템 요소나 서브시스템의 결함은 시스템 실패와 직결되기 때문에 시스템의 실패율은 시스템 요소의 결함율과 비례한다. 규모가 작은 시스템의 신뢰도를 측정하는데는 편리하나 대규모 시스템 특히 결함 허용 시스템의 신뢰도 모델로는 부적합하다.

콤비나토리알 모델 : 이 모델은 부품 계수 모델의 확장으로 결함 트리(fault tree), 성공 트리(success tree), 또는 신뢰도 블럭 도표(reliability block diagram) 등이 있다. 중복 요소가 들어간 간단한 결함 허용 시스템의 신뢰도 모델로는 적합하나 결함 보상 능력이나 시스템 재구성 등은 적절히 표현하기가 어렵다. 그림 2는 성공 트리의 두 가지 예를 보여준다.

상태-공간 모델 : 이 모델에서는 시스템 또는 시스템 요소들의 상태가 해당되는 모델의 상태에 사상(mapping)된다. 따라서 시스템의 신뢰도는 모델의 각 상태에 있을 확률을 구함으로써 여러



(그림 3) 두 프로세서 시스템의 Markov 신뢰도 모델

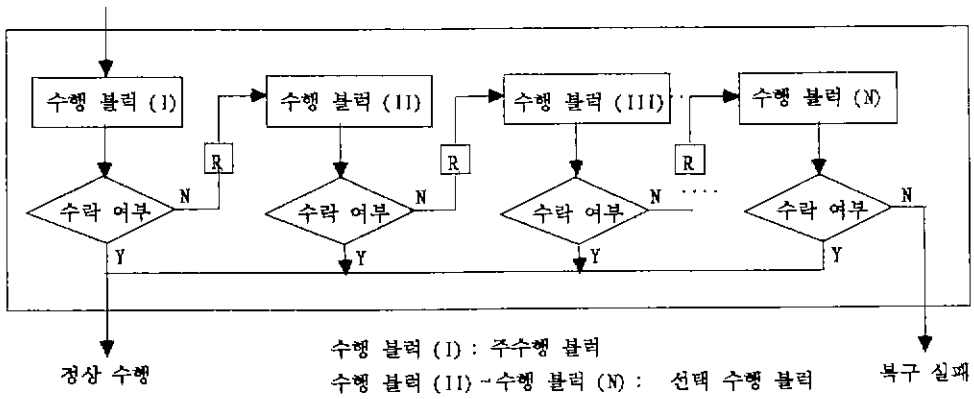
가지 metrics에 대해 쉽게 평가할 수 있다. 예를 들어 MTTF는 "down상태"에 이를때 까지의 평균시간 그리고 유용성은 "down상태"에 있는 시간과 "수행중 상태"에 있는 시간을 비교하면 된다. 상태-공간 모델로는 Markov 사슬 모델이 가장 많이 사용된다. 그림 3a는 Markov 신뢰도 모델의 예로서 두 개의 프로세서에 의한 결함 허용 시스템이다. 그림 3b는 불완전 보상을 고려했을 경우의 모델이다.

2.3 모델에 대한 개량

모델이 개발되면 분석하고자 하는 시스템에 적용시켜 보아 설계 결정에 필요한 충분한 자료를 얻을 수 있는가 점증한다. 만약 부족한 점이 있다면 모델에 대한 개량이 이루어져야 한다. 모델개발시 다음과 같은 여러가지 오류가 발생할 수 있다. 첫째, 시스템에 주어진 여러 가지 조건에 대한 가정을 단순화하는 작업에서 오류가 발생하면 그 결과는 정확치 못하다. 예를 들어 종속성 있는 요소들 간의 관계가 무시된 경우 그 결과는 정확하지 못하다. 둘째, 모델 설계시의 오류, 예를 들어 필요한 상태나 변화의 누락 등이다. 셋째, 모델에 들어가는 파라미터, 예를 들어 실패확률과 수선확률 등의 선정에 오류가 발생하는 경우이다. 많은 경우 이러한 파라미터들은 시스템의 설계가 진행되면서 보완되어야 한다. 그리고 마지막으로 상태 확률의 계산시 계산 알고리즘이나 시뮬레이션 방법에 오류가 생길 수 있다. 이러한 오류가 발생하면 정확치 못한 분석결과를 얻게 되므로 개발된 모델에 대한 철저한 검증이 필요하다. 그리고 수정 보완된 모델은 다시 시스템에 적용시켜 정확한 자료를 얻을 수 있는지 검증하여야 하고 이러한 작업이 반복되면서 보다 좋은 모델을 만들 수 있다.

III. 복구 블럭의 신뢰도 모델

복구 블럭은 주 수행 블럭과 한개 또는 그 이상의 선택(alternative)수행 블럭, 그리고 수행 블럭의 결과를 검사하여 수락할 수 있는지의 여부를 판정하는 수용 검사 루틴으로 구성되어 있다[12]. 그림 4에서 보는 바와 같이 주 수행 블럭이 먼저 수행되고 수행 결과에 대한 검사를 한다. 그 결과가 수락되면 나머지 선택 수행 블



(그림 4) 복구 블럭

력은 수행할 필요가 없다. 그렇지 않으면(즉 주 수행 블럭의 결과가 수락되지 않으면) 상태 복구가 이루어진 후 첫번째 선택 수행 블럭이 수행되고 그 결과에 대한 수락여부를 정한다. 만약 수락이 안되면 그 다음 선택 수정 블럭이 수행되고 같은 작업이 반복된다. 본 장에서는 결함 트리를 이용한 신뢰도 모델과 Markov 사슬을 이용한 신뢰도 모델을 소개한다.

3.1 결함 트리를 이용한 신뢰도 모델

복구 블럭의 신뢰도를 구하기 위해서는 우선 각 복구 블럭이 올바른 결과를 낼 확률을 정의해야 한다. 또한 수행 검사 루틴이 100% 정확하다고 볼 수 없으므로 옳은 결과가 수락 안될 경우와 그른 결과가 수락될 경우도 고려하여야 한다. 시스템의 신뢰도는 다음의 확률에 의해 결정된다.

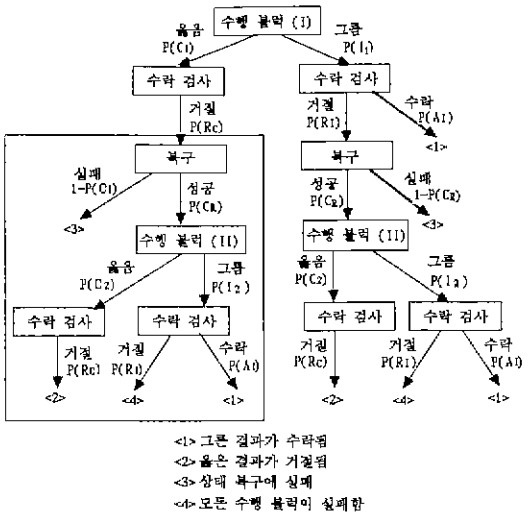
$P(C_i)$=수행 블럭 i가 옳은 결과를 낼 확률
$P(A_c)$=옳은 결과가 수락될 확률
$P(R_c)$=옳은 결과가 거부될 확률(=1−$P(A_c)$)
$P(A_I)$=그른 결과가 수락될 확률
$P(R_I)$=그른 결과가 거부될 확률(=1−$P(A_I)$)
$P(C_R)$=상태 복구에 실패할 확률

[14]에서는 복구 블럭의 수행시 나타날 수 있는 오류를 네 가지로 분류하였다.

오류형태 1 : 수행 블럭의 그른 결과가 수락된 경우
오류형태 2 : 마지막 수행 블럭이 옳은 결과를 내었음에도 거부된 경우
오류형태 3 : 다음 수행 블럭의 수행에 실패한 경우, 즉 상태 복구에 실패한 경우
오류형태 4 : 모든 수행 블럭이 그른 결과를 낸 경우

그림 5는 두개의 수행 블럭을 갖은 결함 복구 블럭의 결함 트리로서 위의 네 가지 형태의 오류에 대한 확률을 구할 수 있다.

수행 블럭이 세개인 경우는 그림 5의 〈2〉와 〈4〉의 자리를 점선 부분으로 교체하면 된다. 이렇게 함으로써 수행 블럭이 N개인 복구 블럭의 결함 트리를 구성할 수 있고 각 형태의 오류에



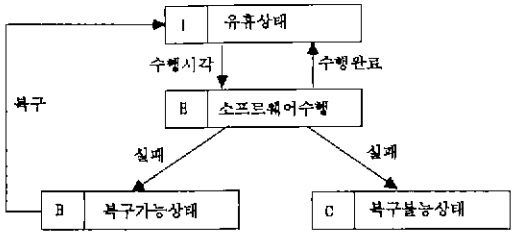
(그림 5) 두개의 수행 블럭이 있을 때의 결함 트리 : 복구 블럭 경우

대한 확률을 구할 수 있다. 이론적으로 N이 커짐에 따라 〈2〉와 〈4〉형태의 오류는 제로에 수렴한다. 만약 수행 블럭 사이에 종속적인 관계가 있다면(예를 들어 공통적인 설계의 결함 등) 주 수행 블럭이 그른 결과를 내었을 경우 선택 수행 블럭이 그른 결과를 낼 확률은 독립적이지 못하다. 이 경우에도 그림 5의 결함 트리는 계속 사용할 수 있지만 각각의 확률은 접합 확률(joint probability)로 구해져야 한다.

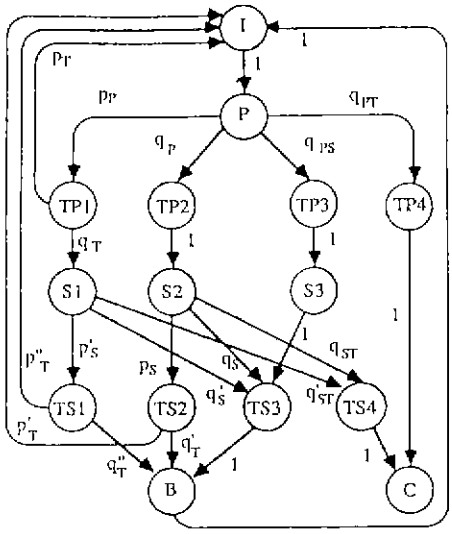
3.2 Markov 사슬을 이용한 신뢰도 모델

[2]에서는 시스템의 움직임을 Markov사슬로써 모델링 하였다. 그림 6은 일반적인 소프트웨어의 수행 모델로서 네개의 상태 즉 I(유휴 상태), E(소프트웨어 수행상태), B(복구가능 상태) 그리고 C(복구불능 상태)를 갖는다. 두개의 소프트웨어 실패 상태(B와 C)가 있는데 B의 경우는 서비스 복구가 가능한 반면 C의 경우는 시스템 복구가 불가능한 상태를 나타낸다. 물론 신뢰도를 계산할 때는 B와 C 두 경우를 모두 고려하여야 한다.

그림 7은 두개의 수행 블럭을 갖는 복구 블럭의 Markov 사슬이다[2]. 상태 I, B, C는 그림 6과 동일하며 그림 6의 소프트웨어 수행 상태(E)는



(그림 6) 소프트웨어 행동 모델



(그림 7) 복구 블럭의 Markov 사슬 신뢰도 모델

P, TPi, Si, TSi로 구성되는데 P는 주 수행 블럭, Si는 선택 수행 블럭, 그리고 TPi와 TSi는 그에 따른 수용 검사 루틴이다. TP1, TP2, TP3, 그리고 TP4는 P의 결과에 따른 상태이며 P에서 TPi로의 변화는 다음의 확률을 갖는다.

TP1 : 결함이 나타나지 않는 경우[p_P]

TP2 : P의 독립적 결함에 의해 오류가 발생한 경우 [q_P]

TP3 : P와 S의 종속적 결함에 의해 오류가 발생한 경우[q_{PS}]

TP4 : P와 수용 검사 루틴의 종속적 결함에 의해 오류가 발생한 경우[q_{PT}]

TP1에서는 I상태로의 변화와 S1상태로의 변화가 가능하다. 즉 주 수행 블럭의 결과가 수용 검사 루틴에 의해 받아들여진 경우 (I로의 변화 [p_T])와 수용 검사 루틴의 오류로 거부되어 (S1로의 변화[q_T]) 선택 수행 블럭을 수행해야 할 경우이다. TP2와 TP3에서도 두개의 변화의 가능성이 있다. 수용 검사 루틴에 의해 그른 결과가 거절된 경우(TP2에서 S2 또는 TP3에서 S3로의 변화)와 수용 검사 루틴의 오류로 그른 결과가 받아들여질 경우도 가능하나 이 경우의 확률은 아주 작기 때문에 고려하지 않았다. 따라서 S2 또는 S3의로의 변화 확률이 1이다. TP4는 수행 블럭과 수행 검사 루틴의 종속적인 결함에 의해 그른 결과가 받아들여진 경우이므로 복구불능의 상태(C)로 변화한다.

주 수행 블럭이 실패하면 선택 수행 블럭이 수행되는데 (S1, S2 또는 S3의 상태) 그 결과에 따라 TS1~TS4의 상태로 변화한다. S1과 S2에서는 P에서의 변화와 거의 동일하다. 그러나 S3는 주 수행 블럭과 종속 수행 블럭의 종속적인 결함에 의한 것이므로 종속 수행 블럭 또한 실패하는 것으로 가정할 수 있다. 따라서 S3에서 TP3에로의 확률이 1이다. TS1과 TS2는 선택 수행 블럭의 결함이 나타나지 않는 경우로서 I(즉 성공적인 복구 블럭의 수행)와 B(복구가능의 결함 상태)의 변화가 가능하다. 그러나 TP3에서는 B로만 그리고 TS4에서는 C로의 변화만이 가능하다. 이와 더불어 그림 7의 모델에서 다음과 같은 가정을 할 수 있다.

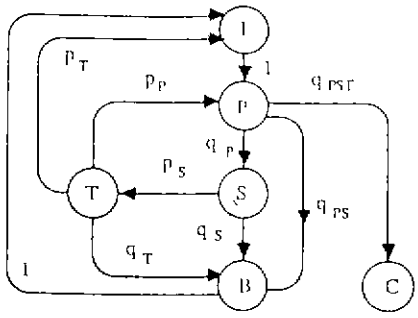
(1) $q_T \approx q'_T$, 즉 수행 블럭의 옳은 결과가 거절될 확률과 선택 수행 블럭의 옳은 결과가 거절될 확률이 비슷하다.

(2) $q_S \approx q'_S$와 $q_{ST} \approx q'_{ST}$, 즉 선택 수행 블럭이 그른 결과를 낼 확률은 주 수행 블럭이 결함을 나타내지 않은 경우나 독립적 결함에 의해 오류가 발생한 경우나 비슷하다.

(3) $P_T'' \ll 1$, 즉 수용 검사 루틴이 주 수행 블럭의 옳은 결과를 안 받아들인 후 선택 수행 블럭의 옳은 결과를 받아들일 확률은 거의 없다. 그리고 한 상태에서 다음 상태로 가는 확률이 1이면 다음 상태와 합칠 수 있다(I, B, C 제외). 따라서 그림 7의 모델은 그림 8의 모델로 단순화될 수 있고 상태 B와 C의 정상 상태의(steady-state) 확률로써 신뢰도를 구할 수 있다.

3.3 토의

본 장에서는 복구 블럭의 신뢰도 모델링 방



(그림 8) 그림 7의 단순화된 Markov 사슬 모델

법으로 콤비나토리알 모델과 Markov 사슬 모델에 의한 방법에 대해 알아 보았다. 두 모델에서 고려한 신뢰도에 직접적인 영향을 미치는 요소는 다음과 같다.

(1) 각 수행 블럭의 신뢰도,

(2) 수용 검사 루틴의 신뢰도, 그리고

(3) 종속적인 결함, 즉 각 수행 블럭 사이의 종속적인 결함과 수행 블럭과 수행 검사 루틴 사이의 종속적인 결함에 의한 오류 발생 확률이다.

이에 따라 결함 트리에서는 네 가지의 오류 형태로 구분되었고 Markov 사슬 모델에서는 크게 두가지의 실패 형태로 나누었으나 그 내용은 거의 동일하다. 다만 Markov 사슬 모델에서는 발생 가능성이 아주 작은 사건은 모델의 간소화를 위해 생략되었고 선택 수행 블럭의 수행을 위한 상태 복구에 대한 것이 별도로 나타나 있지 않다. 반면에 Markov 사슬 모델은 시스템의 상태 특히 종속적인 결함에 의한 오류의 발생상태를 명확히 보여 주며 각 상태의 정상 상태 확률이 구해지면 신뢰도에 관계된 여러가지 metrics를 쉽게 구할 수 있다.

여기서 소개된 것 이외에도 복구 블럭(또는 그의 변형)에 대한 신뢰도 모델에 대한 연구가 있다. 예를 들어 [11]에서는 Markov 사슬을 이용하여 복구 블럭을 모델링하였다. 본 장에서 소개한 모델에서는 결함의 종류, 즉 독립적인 결함과 종속적인 결함에 따라 상태가 정의되었다. 그러나 [11]에서는 수행 블럭의 실패 이유, 즉 그른 결과가 거절된 경우와 옳은 결과가 거절된 경우에 따라 상태가 정의되었다. 또한 3개의 종착 상태(absorbing state)가 있다. 탐지 안된 오류의 발생, 복구 블럭의 결함에 의한 실패, 그리고 수용 검사 루틴의 결함에 의한 실패이다. 따라서 실패의 구분은 본 장에서 소개한 결함 트리 모델과 동일하다. [3,7,9]에서도 소프트웨어의 종속적인 결함의 신뢰도 모델에 대한 연구가 있었다. 이러한 모델들에 대한 결함 허용 소프트웨어의 신뢰도 모델로서의 적합성과 정확성에 대한 비교 연구가 필요하다.

IV. NVP의 신뢰도 모델

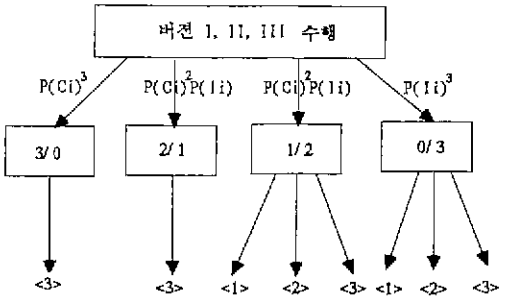
NVP(N-Version Programming)에서는 N개의 서로 다른 버전의 프로그램이 수행된 후 그 결과를 놓고 투표(voting)를 한다[1]. 이때 대다수가 같은 결과를 산출하였으며 투표는 성공하게 되고 그 결과는 옳은 결과로 간주된다. 투표에는 두가지 방법이 있는데 오차를 전혀 허용하지 않는 방법과 오차의 범위를 허용하는 방법이다. 첫번째 방법은 투표방법 자체는 간단하나 버젼이 다른 소프트웨어의 경우 조금씩은 틀리지만 옳은 결과를 낼 경우가 많기 때문에 응용범위가 좁다. 두번째 방법은 첫번째 방법보다 실용적이지만 오차의 범위를 정하는 문제가 간단치 않고 이를 잘못 정하면 그릇된 결과를 옳다고 판정할 수도 있다. 여기서는 두번째 방법이 사용되었다고 가정하였다. NVP의 버젼들은 복구 블럭의 수행 블럭에 대응하며 투표는 수용 검사 루틴에 대응한다. 일반적으로 투표방법이 수용 검사 루틴보다 간단하고 오류를 범할 확률이 작기 때문에 NVP가 보다 실용적이라 할 수 있다. 그러나 최소한 세개 이상의 버젼이 필요하며 최소한 두개 이상의 버젼을 수행시켜야만 투표가 가능하다. 또한 두개 이상의 결과를 비교하는 것(즉 투표) 이외의 결함 탐지 능력이 없다.

4.1 결함 트리를 이용한 신뢰도 모델

NVP의 경우 다음의 세가지 오류 형태가 있을 수 있다[14].

오류형태 1 : 결과가 서로 일치하지 않는 경우;

오류형태 2 : 투표에는 성공하였으나 그 다수



(그림 9) 세개의 버전이 있을 때의 결함 트리 : NVP경우

가 틀린 결과인 경우;

오류형태 3 : 투표 자체에 오류가 있는 경우;

그림 9는 세개의 버전이 있는 경우의 결함 트리이다. 실질적으로 〈3〉형태의 오류는 거의 발생할 확률이 없다. 그리고 각각의 버전이 독립적으로 만들어 졌다면 〈2〉형태의 오류도 제로에 가깝다. 따라서 〈1〉형태의 오류가 가장 일반적이다. 그러나, 각 버전이 독립적이지 못하면 〈2〉형태의 오류도 고려해야 한다.

4.2 Markov 사슬을 이용한 신뢰도 모델

세개의 버전이 있는 NVP는 그림 10과 같이 모델링 할 수 있다[2]. 즉, 세개의 버전 수행후 다섯가지의 상태가 가능하다.

D1 : 결함이 없어 세개의 옳은 결과를 낸 경우 [p]

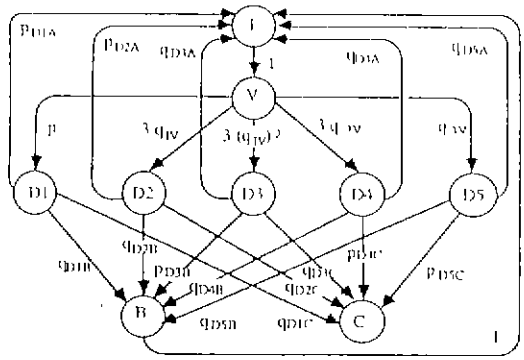
D2 : 한개의 버전에 독립적인 결함이 있어 두개의 버전이 옳은 결과를 낸 경우[$3q_{IV}$]

D3 : 두개 또는 세개의 버전에 결함이 있어 그 결과가 서로 다른 경우[$3(q_{IV})^2$]

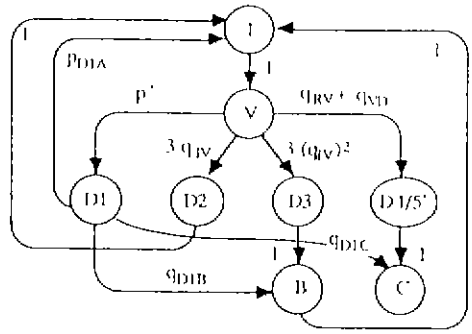
D4 : 두개의 버전에 연관된 오류가 있어 두개의 동일한 틀린 결과가 나온 경우[$3q_{2V}$]

D5 : 세개의 버전에 연관된 오류가 있어 세개의 동일한 틀린 결과가 나온 경우[q_{3V}]

만약 투표에 이상이 없다면 D1과 D2는 I상태로, D3는 B 상태로, 그리고 D4와 D5는 C상태로 가게 된다. 그러나 투표시의 결함에 의해 비록



(그림 10) NVP의 Markov 사슬 신뢰도 모델



(그림 11) 그림 10의 단순화된 Markov 사슬 모델

그 가능성은 희박하지만 다음의 일들이 발생할 수 있다.

(1) 오류 보상 : 다수의 버전이 그른 결과를 산출하였더라도 투표시 옳은 결과가 산출되는 것, 즉 D3, D4, 그리고 D5에서 I로의 변화;

(2) 수행 거절 : 다수의 의견이 거절되는 경우, 즉 D1, D2, D4 그리고 D5에서 B로의 변화; 그리고

(3) 오류 산출 : 틀린 결과가 투표에 의해 산출되는 것, 즉 D1, D2, D4 그리고 D5에서 C로의 변화이다.

따라서 이론적으로 D1에서 D5까지의 모든 상태에서 I, B, C의 모든 상태로의 변화가 가능하다. 그림 10의 모델에서 다음과 같은 가정을 할 수 있다.

(1) $q_{D1C} \approx q_{D2C}$, 즉 D1과 D2에서 투표의 이상으로 복구 불능 상태(C)가 되는 확률은 비슷하다.

(2) $q_{D1C} \approx q_{D5C}$, 즉 D4와 D5에서 투표의 이상

으로 복구 불능 상태(C)가 되는 확률은 비슷하다.

그림 10은 그림 11로 단순화되며 상태 B와 C의 정상 상태의 확률을 구함으로써 신뢰도를 측정할 수 있다.

4.3 토의

본 장에서 소개한 결함 트리에 의한 NVP 모델에서는 세개의 오류 형태를 정의하였다. 반면 Markov 사슬 모델에서는 세개의 버젼 수행 후 다섯가지 가능한 상태가 있으며 투표의 결과에 따라 여러 형태의 오류가 발생하는데 크게 복구 가능의 상태와 복구 불능의 상태로 나누었다. 시스템에 따라서는 두개의 실패 상태로의 구별이 의미가 없을 수도 있으나 안전도(safety)를 예측할 때는 복구 가능 상태는 제외된다. 즉 안전도의 계산에서는 그른 결과가 받아들여진 경우만 고려된다.

NVP의 신뢰도 모델링은 복구 블럭의 신뢰도 모델링에 비해 일반적으로 단순하다. 그 이유는 복구 블럭과 같은 수행 블럭의 결과를 놓고 매번 검사하는 것이 아니라 세개의 버젼의 결과를 갖고 한번의 투표만을 하기 때문이다. 또한 투표에 의한 방법이 복구 블럭의 수용 검사보다 간단하다. 특히 오류를 허용하지 않는 투표일 때는 투표 자체에 의한 오류는 거의 없다고 할 수 있다. 이러한 신뢰도 모델에 근거하여 복구 블럭과 NVP의 장단점에 대한 비교 연구 또한 필요하다. [14]에서는 복구 블럭과 NVP의 결합된 방법에 대한 모델링 연구도 있다. Consensus 복구 블럭이라 불리우는 이 방법은 투표에 의해 다수 결과가 없을 경우 수용 검사 루틴을 사용하여 각 수행 블럭의 수락 여부를 검사한다. 따라서 복구 가능 실패 상태 (B)의 확률을 줄일 수 있으나 복구 불능 실패 상태 (C)는 비록 그 확률은 작지만 증가한다.

V. 결론 및 앞으로의 연구 방향

차 세대의 컴퓨터에 있어서 소프트웨어의 중요성에 대해서는 다시 언급할 필요가 없다. 그러나 높은 개발 비용과 시간, 그리고 무엇보다 신뢰도 높은 소프트웨어의 개발 등 아직도 많은 문제가 있다. 소프트웨어의 공학적인 면에서의 결함을 줄이기 위한 많은 연구와 더불어 개발 또는 설계된 소프트웨어의 신뢰도 측정에 대한 연구 또한 매우 중요하다. 신뢰도 측정은 크게 두가지 목적이 있다. 첫째는 신뢰도 예측을 함으로써 요소에 대한 신뢰도 할당, 그리고 둘째는 만들어진 시스템의 신뢰도를 측정함으로써 시스템 또는 시스템 요소들의 재설계이다. 이를 위한 여러 연구가 있었으나 아직도 미흡하다. 시스템 레벨에서의 신뢰도를 구하는 방법과 더불어 각 요소별의(특히 소프트웨어 요소의) 신뢰도를 구하는 방법이 개발되어야 한다. 앞으로 분산/병렬 처리 소프트웨어 그리고 실시간 시스템 등 소프트웨어의 구조가 점점 더 복잡해짐에 따라 이에 대한 연구 또한 필요하다. 따라서 앞으로 다음의 방면에 보다 많은 연구가 있어야겠다.

- 시스템의 특성에 맞는 모델링 기법의 개발
- 소프트웨어 요소 자체의 신뢰도 예측 기술 개발
- 소프트웨어와 하드웨어 요소의 결합된 신뢰도 모델 기법 개발
- 분산 시스템을 위한 신뢰도 모델 개발
- 실시간 시스템에서 시간 결함(time fault)을 고려한 신뢰도 모델 개발

마지막으로 모델링 방법에 대한 연구와 더불어 소프트웨어 결함 허용 기법들의 효율성, 즉 개발 비용과 그에 다른 신뢰도의 증분에 대한 이론적인 연구와 보다 많은 실험적인 연구가 필요하다.

참 고 문 헌

1. T. Anderson, *et. al.*, "Software Fault-Tolerance: An Evaluation", IEEE Trans. on Software Engineering, Vol. 11, No. 12, December 1985, pp. 1502~1510.
2. J. Arlat, K. Kanoun, and J-C. Laprie, "Dependability Modeling and Evaluation of Software Fault-Tolerant Systems", IEEE trans. on Computer, April 1990, pp. 504~513.
3. D. E. Eckhardt and L. D. Lee, "A Theoretical Basis for the Analysis of Multiversion

Software Subject to Coincident Errors", IEEE Trans. on Software Engineering, Vol. SE-11, No. 12, Dec. 1985, pp. 1511~1517.
4. R. Geist and K. Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques", IEEE Computer, July 1990, pp. 52~61.
5. K. H. Kim, and S. M. Yang, "Fault Tolerance Mechanisms in Real-Time Distributed Operating Systems: An Overview," Proc. PCCS 85, Oct. 1985, pp, 220~229.
6. H. Kopetz, Software Reliability, Springer-Verlag, 1979.
7. B. Littlewood, "Software Reliability Model for Modular Program Structure", IEEE Trans. on Reliability, Vol. R-22, No. 3, Aug 1985, pp. 241~246.
8. J. D. Musa and W. W. Everett, "Software-Reliability Engineering: Technology for the 1990s", IEEE Software, Nov. 1990, pp. 36~43.
9. V. F. Nicola and A. Goyal, "Modeling of Correlated Failures and Community Error Recovery in Multiversion Software", IEEE Trans. on Software Engineering, March 1990, pp. 350~359.
10. J. S. Ostroff, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems", The Journal of Systems and Software, April 1992, pp. 33~60.
11. G. Pucci, "A New Approach to the Modeling of Recovery Block Structures", IEEE Trans. on Software Engineering, Vol. 18, No. 2, February 1992, pp. 159~167.
12. B. Randell, "System Structure for Software Fault Tolerance," IEEE Trans. on Software Eng., June 1975, pp. 220~232,
13. A. L. Reibman and M. Veeraraghavan, "Reliability Modeling: An Overview for System Designers", IEEE Computer, April 1991, pp. 49~57.
14. R. K. Scott, J. W. Gault, and D. F. McAllister, "Fault-Tolerant Software Reliability Modeling", IEEE Trans. on Software Engineering, May 1987, pp. 582~592.

양 승 민

1978 서울대학교 전자공학과 학사
1978 ~1981 삼성전자(주) 엔지니어
1981 ~1983 University of South Florida MS in Computer Science
1983 ~1986 University of South Florida Ph.D in Computer Science
1986 ~1987 University of South Florida 조교수
1987 ~1992 University of Texas at Arlington 조교수
1993 ~현재 숭실대학교 소프트웨이공학과 부교수
관심 분야 : 결함허용 실시간 시스템, 분산처리 시스템, 소프트웨어 공학